Succession: Measuring Transfer of Code and Developer Productivity

Audris Mockus
Avaya Labs Research
233 Mt Airy Rd, Basking Ridge, NJ
audris@avaya.com

Abstract

Code ownership transfer or succession is a crucial ingredient in open source code reuse and in offshoring projects. Measuring succession can help understand factors that affect the success of such transfers and suggest ways to make them more efficient. We propose and evaluate several methods to measure succession based on the chronology and traces of developer activities. Based on ten instances of offshoring succession identified through interviews, we find that the best succession measure can accurately pinpoint the most likely mentors. We model the productivity ratio of more than 1000 developer pairs involved in the succession to test conjectures based on the organizational socialization theory and find the ratio to decrease for instances of offshoring and for mentors who have worked primarily on a single project or have transferred ownership for their non-primary project code, thus supporting a theory-based conjectures and providing practical suggestions on how to improve succession.

1. Introduction

Present software development business practices are trying to emulate the success of manufacturing process by offshoring software development to countries with lower labor costs and higher availability of workers. The relatively more complex domain of software development is making it difficult to achieve cost savings comparable to offshored manufacturing. In this work we investigate possible reasons for that challenge. Our primary goal is to create methods to identify instances of succession under the assumption that simpler and direct but intrusive approaches, such as interviews, are too costly, impractical, or impossible in many real-life scenarios. A second goal is to investigate the effects of succession on developer productivity given that cost-reductions is a common reason to implement succession.

At the conceptual level, the code ownership transfer, among other impacts, may lead to a change in the organizational structure without the corresponding change in the product structure. Therefore, we expect to see some changes in the product structure (or a change in the structure of the receiving organization) as a result of such transfers. Finding such organizational and product structure evolution may provide insights and recommendations related to code ownership transfer in particular and for improvements of organizational and technical structure in general. More generally, such transfers are manifest examples of organizational socialization [20] and, according to that theory, are crucial ways in which organizational knowledge and culture related to software development is preserved.

Our analysis primarily focuses on ramifications of code ownership transfer for commercial and open source software. However, we validate our succession measures in the offshoring context, therefore some aspects of the findings are likely to be specific to that context. We chose offshoring succession for validation because of its significance to business and society and because offshoring succession is often easier to verify due to the deep emotional and practical impact it often imprints on the participants. Succession within organizations is often quite informal, is less readily recognized by participants, and, therefore, is much harder to be unequivocally established in an empirical investigation.

Given the difficulty of defining and measuring succession, we are focused on reframing the concepts to reflect the same or similar phenomena *and* be subject to measurement, and on techniques that demonstrate its impact on products and organization. The first concept involves *implicit* or *virtual teams* and it represents undirected relationships among individuals based on the affinity to the parts of the product they are working or have worked on. Implicit team members may know each other and communicate if they are working on the same part of the product at the same time. However, if

they are separated temporally or are working on cloned versions of the product, they may be unaware of each other's existence. Thus, they may not form a team in the ordinary sense of the word and, consequently, we use the term *implicit team.*

The second concept is needed to define the transfer of responsibilities to maintain and enhance a part of a product and involves directed (often temporal) relationships between individuals within the implicit teams reflecting the chronological order in which different individuals were engaged with (owned) a particular part of a product. We call it *succession*[1]. There are various types of succession: we are primarily concerned with offshoring and refer to receiving party as *followers* and to the transferring party as *mentors.*

Our first objective is to create measures of implicit teams and of the succession in them. The second objective of this work is to relate succession to outcomes that motivate the practice in the first place: the reduction of development costs. To achieve that, we first develop reliable measures of organizational dynamics that reflect succession. Then, we use these measures to identify multiple instances of succession and to compare mentor and follower productivity under a variety of scenarios. The hypotheses are based on conjectures from the theory of organizational socialization [20] that investigates how individuals learn established organizational practices and values.

We start from a description of the approach we took in Section 2 and continue with the description of the context for our study in Section 3. The measurement framework for succession is described in Section 4 and evaluated in Section 5. Section 6 evaluates the impact of succession on developer productivity, related work is discussed in Section 7, and conclusions are presented in Section 8.

2. Methodology

Briefly, the overall approach we take to investigate succession is to start from the general assumption that such phenomena leave traces in the development and IT systems and formulate hypotheses that reflect our experience and intuition about how such phenomena may be reflected in the observable traces and digital artefacts in software change and problem reporting data. We then refine and validate these ideas using a small empirical study that identifies actual instances of succession. The hypotheses are evaluated by observing if the observed patterns fit these actual instances of succession. We then apply these patterns to determine succession on other parts of the product and in other projects and investigate how productivity of developers changes as a result of the succession.

Notably, the overall approach we take is quite similar to archaeology, except for the fact that we study a narrow aspects of human culture reflected in organizational relationships and *recover*, *document*, and *analyze* digital (not material) remains. Because observations available to us involve only projections of the organizational structure on the work product and support systems, the attempt to reconstruct organizational structure is analogous to image tomography (image, often three-dimensional, reconstruction from multiple projections), thus the term organizational tomography may be suitable to describe methods reconstructing organizational structure from the projections or traces in version control, problem tracking, and other supporting systems. The amount and complexity of available data in such systems necessitates the use of analysis tools except, possibly, in the smallest projects. Therefore, we rely on methods described in [11]. The following steps are applied iteratively until data of sufficient quality to perform desired analysis is obtained:
1. Retrieve the raw data from the underlying systems via access to the database used in the project support tools or by "scraping" relevant information from the web interfaces of these systems.
2. Clean and process raw data to remove artifacts introduced by underlying systems. Verify completeness and validity of extracted attributes by cross-matching information obtained from separate systems.
3. Construct meaningful measures that can be used to assess and model various aspects of software projects.
4. Analyze data and present results and collect feedback for further validation.

In this project we rely on data that has passed through the first two levels of the pipeline and we will focus primarily on the elaboration of the remaining two steps. To discover measures of virtual teams and succession we propose hypotheses that state how such phenomena would be reflected in the observable traces as software changes. These hypotheses are used to construct measures that represent features of interest (step 3 in the overall analysis). The succession measure defined on the domain of developer pairs can be thought of as a likelihood function indicating probability that the first developer has taken over some (or all) of the responsibilities of the second developer. The pairs with the highest likelihood can then be expected to represent instances of succession.

To investigate these hypotheses and to validate the

[1]We borrow the meaning from ecology where succession means the gradual and orderly process of change in an ecosystem brought about by the progressive replacement of one community by another until a stable climax is established.

succession we conduct an empirical study to identify actual instances of the phenomena through interviews and document search. The family of proposed measures is evaluated on this sample to determine how well they capture the phenomena of interest. We do that by calculating how the actual mentors ranked with respect to other developers according to the succession measure.

Once we identify the most suitable succession measure, we use it to identify succession on a larger sample of developers where we can not use interviews due to costs or inability to contact relevant people who, in many cases, have left the project. This larger and more diverse sample of follower-mentor pairs is used to study follower/mentor productivity ratio. To pose hypothesis in this domain we rely on the organizational socialization theory.

3. Context

We investigate software development in Avaya with many past and present projects of various sizes and types involving more than two thousand developers. As described above, we conduct two empirical studies: one involving the validation of succession measures and another involving modeling of the productivity ratio. In the first study we identified actual instances of succession in a medium sized (1-3 MNCSL) project that has used offshoring practices for several years and has built a substantial expertise in offshoring. In broad strokes, the practice identifies the tasks and individuals (mentors) whose work is a candidate for offshoring and obtains their cooperation by assigning them different responsibilities or by providing a separation bonus contingent on expertise transfer. At the same time, a small team of developers in the location is identified and their team lead is brought to the mentor's location to follow (shadow) mentor's work by participating in meetings, phone calls, and other business related activities together with the mentor. After a few weeks of shadowing, the team lead returns to the offshore location and trains remaining members of the team. Even though the commonly used term is *shadow*, we do not believe it properly reflects the semantics of what is going on[2]. We, therefore, propose to use the term *follower*, because it captures the aspect of *shadowing* by following the mentor around, and the aspect of learning: one that follows the opinions or teachings of another.

In particular, we have identified 14 mentors and their followers to evaluate measures of code ownership transfer. Four of these individuals were not involved in development tasks, therefore we had only ten pairs representing succession of development work.

The second study involves more than one thousand followers for thirteen major products ranging from 29 to 252 developers residing in five US and five international locations ranging in size from 12 to 482 followers with the primary offshore location having 182 followers. The remaining international locations were either no longer a destination for offshoring work or were there because of a prior acquisition of another company. This set of followers was selected from a much larger set of all Avaya developers and software projects to exclude numerous smaller, not affected by offshoring, or no longer active projects.

Two primary sources of data were utilized in the study. The changes to the source code were obtained from a variety of version control systems used in Avaya, including SCCS, ClearCase, CVS, and SubVersion. The data were cleaned to eliminate administrative changes (changes made for the purpose other than to enhance or fix the product) using a variety of techniques appropriate for each system and each project. For example, the branch delta in SCCS or CearCase that do not make changes to the underlying source code were excluded. The data cleaning and validation was done to support project measurement and prior studies, for example, [8, 13] and, therefore is not described in more detail here. We used developer login making the change, the date of change, and the filename (including path) of the changed source code file. Because several projects have changed source code repositories over the considered period, we have normalized the pathnames of files in such projects to be independent of the repository. Furthermore, we have mapped the pathnames to product names to associate each file with a software product where it was used.

The second source of data was an organizational database (POST) that lists individuals, their organization, and contact information. We had collected frequent snapshots of this data over a period of seven years. The purpose of this data was to establish the location(s) for each developer. As any other source of data, it had its share of anomalies and issues. First, developer logins were not always identical to email handles in POST. Furthermore, logins have changed over time for some developers because a recent policy required logins to match email handles. Third, the email handles and even organizational IDs have changed for some developers, especially for a small group of developers offshore that were initially brought to the US location and later went to their permanent offshore location where they got a new organizational ID. To

[2]For example, the mentor would need to be called *obscurer* even though she *enlightens* the shadow with her expertise.

deal with these issues we used a NIS database (snapshots of which we have also collected over seven years) that mapped login to the organizational ID and the full name of the person authorized to use the login. This extra piece of information allowed us to establish the identities of developers over time despite the organizational ID's, email handles, and sometimes even names (for example, as a result of a marriage) changing. We have used these sources of data to map logins and organizational IDs to unique numeric IDs identifying each participant. These unique IDs were then substituted for logins in the code change data and for organization IDs in the POST data to normalize identity information and to provide more privacy (some developers could be recognized from their login). From POST data we used only the location of the developer based on their address and phone number. Obviously, some developers have changed locations over time. We have excluded few developers that moved across non-offshore locations from further analysis and associated only the main offshore location for the followers that spent some time early on in the location where they were mentored.

4. Measurement

To infer code transfer patterns from version control data we postulated four intuitive measures of succession and ranked all present and former developers based on how close they were to each follower. The rank of the real mentor would be high if the measure approximates the likelihood of succession. To define these measures we consider how the succession may be reflected/projected onto the development support systems. First, the responsibilities to maintain and enhance the code leave records of code changes in a version control system. Second, the chronological order of engagements by *mentors* and *followers* should be reflected in the temporal order of these changes. The reconstruction or tomography problem is then to reconstruct implicit teams and succession from such change records. Implicit teams may then be measured by linking developers that are changing the same packages, files, methods, or even lines of code, for example, by counting the number of files both developers have changed in the past. Succession may be measured by selecting pairs of developers with the most clear succession signature reflecting the location in the code and chronological order of the changes. The measures of succession were constructed so that for developer a the mentor b is determined by finding the developer maximizing the succession measure:

$$b = \arg \max_{b \in \{Developers\}} S(a, b) \tag{1}$$

Denote files as $f_i,\ i = 1, \ldots, N$, developers as $d_j, j = 1, \ldots, M$, and the time of changes as $c_k(f_i, d_j), k = 1, \ldots, K_{ij}$. The idea behind the first measure is to capture the temporal aspect of succession when one developer changes the file after another developer. The first measure S_0 counts files[3] where the first[4] change a developer d_{j_0} made occurred after the first change a developer d_{j_1} made. Denote the time of such first change as $FC(f_i, d_j) = \min_k c_k(f_i, d_j)$. The first measure of succession is the cardinality of the subset of files both developers changed, but developer d_{j_0} made the first change later than developer d_{j_1}:

$$S_0(d_{j_0}, d_{j_1}) = \aleph\{f_i : FC(f_i, d_{j_0}) > FC(f_i, d_{j_1})\}$$

This measure treats all files equally, however some files may be more relevant to the succession.

The idea behind the second measure S_1 is to take the temporal aspects of S_0 into account and weights each file by the fraction of changes developers made on that file so that the files a developer changes most frequently get more weight. Denote the number of changes developer j made to file i as n_{ij}, then:

$$S_1(d_{j_0}, d_{j_1}) = \sum_{i: \left\{ \begin{smallmatrix} n_{ij_0}, n_{ij_1} > 0 \\ FC(f_i, d_{j_0}) > FC(f_i, d_{j_1}) \end{smallmatrix} \right.} \left(\frac{n_{ij_0}}{\sum_l n_{lj_0}} + \frac{n_{ij_1}}{\sum_l n_{lj_1}} \right).$$

This way the files central to each developer get more weight and have a large effect on the overall measure. If developers overlap only on files they tend to change infrequently, the measure S_1 would be low. The measure may take values in the interval $[0, 2]$, with $S_1 = 0$ indicating no overlap in files that were first touched later by developer j_0 and with $S_1 = 2$ indicating that developers changed the same files with developer j_0 always making later first change than developer j_1.

The third measure S_2 also combines aspects of succession and implicit teams, but this time the weight is based on the relative number of changes the two developers made to a file. Files changed mostly by others where the two developers had contributed little would not contribute much to the measure, but files where at least one developer made significant fraction of changes

[3]For finer or coarser granularity it may make sense to count individual lines, methods, or packages.

[4]We also considered median and last changes, for all four measures of succession, but they did not perform well identifying succession.

would contribute a lot.

$$S_2(d_{j_0}, d_{j_1}) = \sum_{i:\left\{ \begin{smallmatrix} n_{ij_0}, n_{ij_1} > 0 \\ FC(f_i, d_{j_0}) > FC(f_i, d_{j_1}) \end{smallmatrix} \right.} \frac{n_{ij_0} + n_{ij_1}}{\sum_j n_{ij}}$$

S_2 also ranges from zero to two, with the value zero indicating no overlap and value two indicating perfect overlap as in measure S_1.

The fourth measure S_3 combines aspects of all three measures by weighting by the frequency a file was modified by a particular developer and by the fraction of developer's changes that are devoted to a file:

$$S_3(d_{j_0}, d_{j_1}) = \sum_{i:\left\{ \begin{smallmatrix} n_{ij_0}, n_{ij_1} > 0 \\ FC(f_i, d_{j_0}) > FC(f_i, d_{j_1}) \end{smallmatrix} \right.} \frac{\frac{n_{ij_0}^2}{\sum_l n_{lj_0}} + \frac{n_{ij_1}^2}{\sum_l n_{lj_1}}}{\sum_j n_{ij}}.$$

Measures $S_i, i = 1, 2, 3$ would be symmetric if the condition $FC(f_i, d_{j_0}) > FC(f_i, d_{j_1})$ was eliminated, making them suitable to measure implicit teams, not just code transfer phenomena.

Figure 1 illustrates the measures on a trivial example of two developers, two files and six changes. Patterns represent files and colors represent developers with squares representing changes. Curves link developers to files for each change. Because $S_0(d_1, d_2) = S_0(d_2, d_1) = 1$ the first measure can not identify which developer is a mentor. The second measure indicates that d_2 is a mentor for d_1: $S_1(d_1, d_2) = \frac{1}{2} + \frac{2}{3} = \frac{7}{6}$ and $S_1(d_2, d_1) = \frac{1}{2} + \frac{1}{3} = \frac{5}{6}$. The third and fourth measures show the opposite, that d_1 is a mentor for d_2: $S_2(d_1, d_2) = \frac{1}{4} + \frac{2}{4} = \frac{3}{4}$, $S_2(d_2, d_1) = \frac{1}{2} + \frac{1}{2} = 1$, $S_3(d_1, d_2) = \frac{\frac{1^2}{2} + \frac{2^2}{3}}{4} = \frac{11}{24}$, $S_3(d_2, d_1) = \frac{\frac{1^2}{2} + \frac{1^2}{3}}{2} = \frac{5}{12}$.

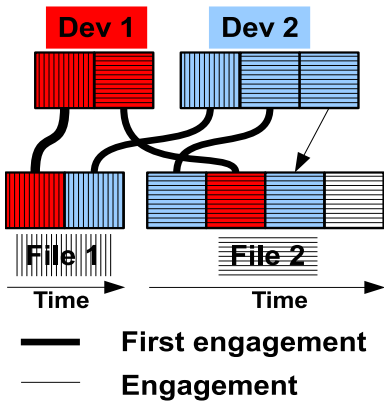

Figure 1. Illustration of how succession measures are calculated.

5. Evaluation of succession measures

To evaluate these four measures we need to considered how close the best fitting mentor defined by Equation 1 is to the actual mentor. For each measure S_i and each follower d_j we ordered all remaining developers d_k according to the magnitude of $S_i(d_j, d_k)$ in decreasing magnitude, so that $k_0 = \arg\max_k S_i(d_j, d_k)$, $k_1 = \arg\max_{k \neq k_0} S_i(d_j, d_k)$, and so on. For each follower we thus got a list of values for each measure that was sorted by magnitude in decreasing order. That way each follower got a list of all potential mentors ordered by a particular measure. Looking at a particular follower and a particular measure the first developer in this ordered list represents the best mentor (according to that measure) for that follower. We then looked at the rank (position in this ordered list) of the actual mentor. These ranks (starting from zero) are presented in Table 1. In other words, if d_{k_0} is the actual mentor, then Table 1 contains zero, if d_{k_1} is the actual mentor, then Table 1 contains one, and so forth.

There are several patterns visible in the table. Surprisingly, the purely temporal first measure S_0 appears to perform quite well in detecting mentor-follower relationships. Another surprise is that, arguably, the most intuitive measure S_1 where the weighting is done according the fraction of developer's changes on a file, has the worst performance. Weighting by the fraction of file's changes made by a developer (S_2) performs uniformly better than the remaining measures. This suggests that succession and ownership are mostly related to the fraction of file's changes performed by a developer, and not based on the fraction of developer's changes performed on a file. In other words, what matters most is who owns the file, not which files a developer spends most of their time changing. This is true to the extent that the measure S_3 incorporating both weights appears to be inferior to measure S_2 that incorporates only one weight.

The second observation concerns several followers (2, 3, 4, and 9) for whom none of the measures have top ranks for the actual mentors. A closer look at these instances reveals that actual mentors were senior developers who have previously transferred ownership of the code taken over by the follower to other developers. These earlier followers occupy the top mentor rankings, leaving the actual mentor further down in the ranking list. We illustrate this point in Figure 2 where Follower 2 is shown connected with to the top two mentor candidates according to the top two values of S_2. The top value of S_2 is represented by a solid edge and the second value with a dashed edge. Edge thickness reflects the value of S_2. Black (dark) edges show the top men-

Follower	1	**2**	**3**	**4**	5	6	7	8	**9**	10	Total
V-Team	127	**158**	**161**	**160**	129	165	162	129	**177**	154	1522
S_0	2	**20**	**11**	**56**	0	9	10	0	**8**	2	118
S_1	0	**51**	**9**	**126**	5	44	81	9	**35**	39	399
S_2	1	**23**	**20**	**19**	2	5	3	0	**9**	0	82
S_3	1	**25**	**7**	**111**	4	4	39	0	**13**	0	204
Total	4	**119**	**57**	**312**	11	48	110	17	**82**	41	803
p-val S_2	0.008	**0.146**	**0.124**	**0.119**	0.016	0.03	0.0185	0.008	**0.051**	0.0061	0.054

Table 1. The ranks (starting from 0) of interview-derived mentors according to four measures for 10 followers.

tor candidates and the orange (light) edges show the top follower candidates. The shortest solid edge (top candidate) path from Follower 2 to the actual mentor has three edges including developer 1 and developer 11 (developer numbers are unrelated to the follower numbers in Table 1).

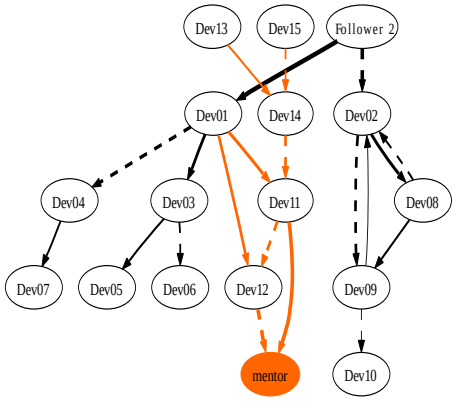

Figure 2. Illustration of top-two mentor candidate graph for Follower 2 and measure S_2.

This suggests that the succession measures may be improved by constructing an ownership transfer graph and determining ownership transfers utilizing properties (shortest paths) of that graph. Other possible improvements may be achieved by considering traces of communication among developers. It is nearly impossible to obtain such communication information based on email or instant messaging for companies operating in countries with strong privacy rules. Fortunately, workflow systems, such as bug tracking systems and discussion boards prevalent in software projects, provide an alternative source of communication patterns.

The best possible sum of the ranks that could be achieved is zero, but it is not realistic given a large number of candidates. The worst possible sum of ranks would be $> 20K$ if the true mentor was ranked last from all developers in the sample. A more reasonable worst-case score would be having the true mentor to be the last among developers that changed at least one file in common with the follower (shown in the row labeled "V-Team" in Table 1). That would lead to a rank sum of 1033, and an average rank sum for a random selection of mentors would be around 500. For comparison, S_2 has a rank sum of 82 (the sum is only 11 if we exclude followers 2, 3, 4, and 9). The comparison with random selection can be formulated as a statistical test calculating the probability that the observed or lower rank could have been obtained purely by chance. These probabilities (commonly known as p-values) are shown in Table 1 for measure S_2 and indicate that measure S_2 is significantly different from a random choice of mentors. Ranks that have their p-value above 0.05 are shown in lighter color in Table 1.

More generally, given that teams of four to five developers are taking over the tasks of an individual, it is not unreasonable to have ranks larger than zero, because code ownership transfer is a group activity. Therefore, the score of 11 for 6 followers (excluding 2, 3, 4, and 9) is probably close to the best we can expect for any measure of succession, while the scores for the remaining followers may be improved through better measures that take into account entire succession graph as suggested above.

6. Evaluating the impact of succession

The ultimate objective of any software engineering investigation is to determine if the phenomena under study has tangible effects on software effort, quality, or lead-time. Because succession is crucial to offshoring practices that tend to be motivated by cost savings, we investigate the impact of succession on developer productivity. To accomplish that we needed to choose a measure of productivity and to choose a larger sample of mentor-follower pairs. To place the investigation in a theoretical framework we chose the theory of or-

ganizational socialization [20] because it can be used to pose testable hypotheses about the outcome of various succession scenarios. We start with conjectures of how different scenarios of succession should affect developer productivity in Section 6.1, select the most suitable measures of productivity in Section 6.2, discuss the sample selection process in Section 6.3, present the resulting model of productivity ratio in Section 6.4, and discuss threats to validity in Section 6.5

6.1. How Succession Affects Developer Productivity

The organizational socialization theory [20] investigates how organizational culture including values, norms, and practices is transferred and assimilated by participants. In our context we are focused on development practices specific to a particular product or a part of product. These practices, to a large extent, include the knowledge about other key players in the project, their roles and responsibilities, and the ways of interaction that are more likely to bring the desired results. Therefore the theory of organizational socialization is a useful tool to analyze succession.

The concept of organization and individuals is based on [17], which includes functional, hierarchical, and interactional dimensions. Functional dimension defines the type of tasks individuals perform. We focused on the developer tasks, so the differentiation was primarily based on the products or parts of products developers were involved in. The hierarchical dimension defines reporting structure and it was primarily reflected in the fact that the line management was primarily location specific. Therefore, for developers spanning location boundary the lowest common supervisor tended to be further up the management hierarchy than in cases of same-site interactions. The interactional dimension defines the centrality of a person in the decision making process, with more experienced people tending to have more impact. Thus, a senior developer making major architectural decisions would be higher in the interactional dimension than a hypothetical junior developer who is entrusted to fix only medium- and low-severity defects.

The organizational socialization theory classifies outcomes of individual's adaptation to organizational culture into roughly two classes. Custodial outcomes represent complete preservation of organizational culture and job function. Innovative outcomes range from expanding the information sources used to make decisions to a redefinition of job's mission. In our context, an example of an innovative outcome would be a developer assigned to fix defects starting to use novel debugging tools (expanding information sources) or trying to influence development process to improve the quality of bug reports or to prevent introduction of defects (changing job's mission). We assume in our analysis that more innovative outcomes in software development will lead to higher developer productivity.

Van Manen and Schiele [20] distinguish collective vs. individual, formal, vs. informal, sequential vs. random, fixed vs. variable, serial vs. disjunctive, and investiture vs. divestiture socializations as major predictors of the socialization outcome and propose how each may lead from custodial (preserving organizational traditions) to innovative (changing the information and mission of the organizational function) outcomes. We use only a subset of conjectures that are most salient to succession and that can be measured in our context.

In particular, we distinguish between succession within a location and succession that has offshoring as its purpose. In both cases the socialization is individual (mentorship) and serial (taking over mentor's responsibilities). It is not clear to what extent we can determine if investiture (no attempt to change individual) and divestiture (certain personal characteristics of an individual are to be stripped) vary with different scenarios of succession. The within-location succession can be characterized as less formal than across-location succession because the follower works in the same organization and does not have to travel for the explicit internship. Within-location succession is more likely to be random and event driven learning (needing mentor help for a particularly vexing defect) learning than a more scripted and sequential offshoring mentorship scenario. Within-location succession may also have a more variable time-table than the trip-duration bounded schedule of the offshoring followers. However, according to interviews, there is an indication of continued professional collaborations between a follower and a mentor even in cases where the mentor leaves the organization suggesting that even offshoring socialization is variable and goal- not schedule-driven. According to conjectures in [20], informal, random, and fixed time table successions are more likely to lead to innovative outcomes. Given clear differences in informal and random aspects and less clear distinction in the time table, we would expect that:

Proposition 1. Offshoring succession leads to less innovation.

The second aspect that we distinguish in succession scenarios relates to the breadth of mentor's expertise. We operationalize it by the percent of changes done on the product most frequently changed by the mentor. Higher value of that measure indicates that mentor's work is primarily concentrated on that single product, while lower values indicate that the mentor had sub-

stantial experience in other products. Such breadth of experience is likely to increase mentor's understanding of what it takes to master a new codebase and, therefore, to improve succession:

Proposition 2. Mentors with expertise dispersed over several products would provide mentorship that leads to more innovation.

The third aspect that varies in succession scenarios is related to what area of expertise is transferred from a mentor to a follower. Mentors tend to be more senior developers that have worked on several products, but if the succession is done on a product that is not the primary area of expertise for the mentor, it may lead to a less effective transition.

Proposition 3. Mentors that transfer expertise of their secondary products would lead to less innovation by the followers.

We also propose that:

Proposition 4. The effectiveness of expertise transfer increases over time as the organization improves its offshoring practices.

The mentors with the largest numbers of followers are likely to be most productive, therefore an average follower would appear less productive in comparison, thus lowering the productivity ratio. Furthermore, mentors with the largest numbers of followers are less likely to spend as much time on mentorship of each follower, potentially reducing the amount of transferred expertise and resulting in less innovative followers.

Proposition 5. Mentors with more followers would have less innovative followers.

Finally, we expect that the productivity ratio would depend on the complexity of the transferred knowledge. This is not explicitly stated in [20], but can be easily derived based on the understanding that bigger and older products have more elaborate rules and traditions and, therefore, require more custodial responses from the newcomers and requires more time from a newcomer to become central enough in the organization to be able to implement their innovations.

Proposition 6. Products with the oldest and largest code bases are likely to have lower productivity ratios.

6.2. Measuring Developer Productivity

Conceptually, the productivity of a developer is the number of product units (output) produced over some unit of effort (inputs). For commercial developers who are employed full-time, the inputs may be roughly approximated by developer time (staff-months) multiplied by salary and other employment costs. However, unlike in manufacturing, in software the product units are typically not well defined. Most commonly used measures of software output lines of code (LOC) or Non-Commentary Lines of Code (NCSL) are easy to obtain but tend to have numerous drawbacks and have to be adjusted for system size, staffing levels, development capability, programming language, the extent of reuse, and type of development activity (see, e.g., [3, 7]). Another commonly used measure of output is Function Points [1]. However, it is more difficult to calculate and was not used in this organization.

Therefore, we chose to use number of changes per staff-month as a pragmatic measure of developer productivity, because it was readily available (similar to NCSL) and has been successfully used in the past [2, 12]. In particular, the study in [2] has investigated a relationship between software features that are sold to customers and the number of changes needed to implement that feature and found a strong relationship between changes and sellable functionality. The summary of developer experience with respect to a part of the system expressed in the number of changes was found to reflect developers' and managers' subjective perceptions of expertise [12]. Furthermore, a study of global development found that it takes more than a year for developers to reach full productivity (measured in changes per month) on a large telecommunication system [14].

It is important to note that changes per staff-month may not be suitable to measure individual's performance in a performance evaluation setting because it is likely to have fairly large variance and is easy to manipulate if it was used for such a purpose. However, it appears to be adequate in situations where it is not used for performance evaluation (and there are no other motivation for developers to make unnecessary changes), and the sample is large enough to reduce the inherently large variances. One of the key assumption here is that the source code is kept strictly under version control, as was the case in our study. Furthermore, because mentors and followers make changes to the same files, the comparisons between them automatically adjusts for the inherent differences in making changes to different applications, using different programming languages, and other code related factors.

6.3. Inferring succession

The resulting data including numeric IDs, locations, dates, and file names was filtered further to remove developers that were primarily involved in supporting version control, build, and test environments and other internal tools that spanned multiple software projects. Finally, followers that spent less than four months making changes to the code or made less than 100 changes

were excluded to limit the impact of learning on our results.

Based on the experiences of fitting mentor-follower relationships described in Section 5, we applied the best performing measure S_2 on a sample of 1012 potential followers to find their most likely mentors. All follower-mentor pairs overlapped in time, suggesting that the followers were aware of mentor's existence even if they did not go through an explicit mentorship relationship.

6.4. Model of Productivity Ratio

To test propositions in Section 6.1 we fit a regression model with response being the ratio of productivity as defined in Section 6.2 between the follower and the mentor. The predictors are:

PR The ratio of follower to mentor productivity. To make the distribution less skewed we have transformed it using logarithmic transformation.

Fr Date of the first change made by the follower to account for the organizational learning over time.

Off An indicator that the mentor-follower relationship was offshoring.

Prd An indicator that the primary mentor expertise was gained for a different product.

Brdth Concentration of mentor's expertise represented by the fraction of mentor's changes for their primary product.

NF The number of followers for a mentor, representing mentor's skill, seniority, and the lack of time to devote to any particular follower. To make the distribution less skewed we have transformed it using logarithmic transformation.

Lrg Indicator of a very large system with tens of MLOC.

Md Indicator of a medium size system with several MLOC.

The model for the regression shown in Table 2 was:

$$\ln(PR) \sim Fr + Off + Prd + Brdth + Lrg + Md + \ln(NF)$$

According to coefficient of the offshoring predictor, Proposition 1 is supported by the data. In particular, offshoring succession roughly halves the productivity ratio ($e^{-0.63} \approx 0.5$) with 95% confidence interval of $[0.42, 0.67]$. Similar result also holds for Proposition 3. Transfering products that are not primary to the mentor also roughly halves the productivity ratio ($e^{-0.68} \approx 0.5$) with 95% confidence interval of $[0.4, 0.64]$.

Proposition 2 is supported by the data and a hypothetical mentor that spends 100% of their changes on one product leads to roughly half ($e^{-1.41/2} \approx 0.5$) of the productivity ratio as compared to a hypothetical mentor that spends only 50% of changes on their primary product. The 95% confidence interval for this ratio is $[0.38, 0.64]$.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	21.85	19.876	1	0.27
Fr	−0.01	0.010	−1	0.31
Off	−0.63	0.125	−5	0.00
Prd	−0.68	0.118	−6	0.00
Brdth	−1.41	0.267	−5	0.00
Lrg	−1.21	0.166	−7	0.00
Md	−0.46	0.099	−5	0.00
$\ln(NF)$	−0.53	0.033	−16	0.00

Table 2. The productivity ratio model with 1012 mentor-follower pairs and Adj-$R^2 = 59$.

There appears to be no support for Proposition 4 that the productivity ratio increases over time: the coefficient is not significantly different from zero.

Proposition 5 is supported and the productivity ratio drops roughly in proportion to the square root (power of 0.53) of the number of followers.

Finally, Proposition 6 is also supported with the largest products having approximately one third ($e^{-1.21} \approx 0.3$) of the productivity ratio of small products with the 95% confidence interval of $[0.21, 0.42]$. Medium sized products have roughly two-thirds $e^{-0.46} \approx 0.63$ the productivity ratio of small products with the confidence interval of $[0.52, 0.77]$.

It is important to note that the model does not imply that the follower productivity is always lower than the mentor productivity. In fact, under optimal conditions when the succession is not offshoring, the product is small, the mentor is transfering expertise of their main product, and the mentor has one follower, the predicted productivity ratio is bigger than one.

The findings confirm a number of theoretical propositions based on organizational socialization theory and, more importantly, provide a method to collect relevant data and test new conjectures related to the transition of development work.

These findings also have a number of important practical implications. First, the offfshoring succession has high costs. While we could not confirm the need for four to five ratio of new offshore developers per mentor for smaller products (the estimate is two to one), it appears that the ratio needs to be even higher (six to one) for the largest products. Therefore, offshoring should always start with smaller, newer projects and cost and other implications should be carefully considered in the very large and/or very old projects.

In addition to choosing the right project, choosing the right mentor may also increase the productivity ratio. The results suggest that followers with mentors having a broader base of expertise that spans more than one product have better productivity ratio than followers with mentors that worked on a single project.

Mentors appear to have trouble transfering their expertise to a large number of followers, therefore even the best mentors should limit the number of their followers.

6.5. Threats to Validity

Broadly, there are questions about the extent to which the results for Avaya projects would generalize to the rest of the industry, questions regarding the ability of the succession measure to detect the mentor for a given follower, and questions related to the particular model used to test hypotheses of organizational socialization.

The set of Avaya projects considered in this analysis is quite diverse and ranges from embedded devices and high-availability server software to desktop applications. This suggests that similar results may be expected in other companies as well.

The ability of the succession measure to point out actual mentors is another important point. The validation was conducted in the most clear-cut situation of offshoring context and may be more difficult to validate in less clear cases for other types of succession. Nevertheless, the fact that the followers primarily work on the same code as mentors, points out the fact that determining the actual formal mentor for a follower (if one was ever assigned) may not be so important. In fact, the measure could be used as a definition of the "virtual mentor." We have investigated the sensitivity of our results to the alternative selection of mentors. In particular, we fitted the model shown in Table 2 replacing the top choices of the mentors with the second, third, fourth, and fifth best choices (according to S_2 measure) for each follower. The same coefficients were significant and had the same sign, but the model fit went down with each subsequent choice from the adjusted R^2 of 0.6 for the top choice to the adjusted R^2 of 0.35, 0.26, 0.25, and 0.22 for the second, third, fourth, and fifth best choices accordingly. This shows that the results are not sensitive to the top five mentor choices.

Given observational nature of the study, there may be other latent variables that explain variation of productivity ratio and of predictors. However, the predictors themselves were not strongly correlated. Only the predictor number of mentors and the predictor of largest projects had a Spearman correlation of 0.66. Remaining correlations were below 0.4.

7. Related work

While there are no published results that are similar to this investigation, it touches upon multiple established areas of software engineering research. From the measurement perspective the interrelationships among parts of a codebase were investigated in depth in [5]. The fact that expert developers can be identified by observing code that they change was exploited in expertise visualization tool [12]. The relationship among developers using workflow and commonly changed codebase was utilized in identifying interdependencies and coordination requirements in, for example, [8, 6]. In [14], the code was chunked into independently changeable pieces of suitable size to fit the capabilities of a particular offshore development location. A detailed case study of what a volunteer has to go through to join two open-source projects is presented in [21]. Succession, on the other hand, has not been previously measured.

The investigations of developer productivity have a long and rich history from early work on cost estimation models [4] to more recent studies [15] and tool-based approaches [18] that help address the issues of how individual developers deal with code understanding: a problem that is a crucial part of succession. Findings from a more inclusive study of how individual developers work in large projects [9] and tools that support awareness through source control system [16] and code annotations [19] would nicely complement our higher level recommendations of mentor selection in succession.

8. Conclusions

We have proposed and validated a method to measure the phenomena of succession based on the information in the version control and organization directory systems, proposed six organizational socialization theory derived hypotheses on how different types of succession affect developer productivity. The analysis of more than one thousand developers involved in more than ten products shows that there are large differences in the productivity ratio. Larger projects, overloaded mentors, and offshoring succession significantly reduce the productivity ratio. Breadth of mentor expertise and succession where mentor's primary poduct is transfered significantly increase the productivity ratio.

The succession becomes more important as the software development increasingly follows in the offshoring

and outsourcing footsteps of the manufacturing and as open-source code reuse in commercial projects becomes more widspread. More generally, the succession is an essential aspect of organizational dynamics in software projects.

Clearly, despite the large size of the study, the results presented in this work would benefit from replication in other environments, yet promise of research in this area is tantalizing. The potential to track the transfer of code ownership in the universe of all software code [10] and the ability to quantify how the product and organization coevolve are likely to provide numerous lessons and may significantly improve the way software development organizations and product are created and structured in the future.

Acknowledgments

We thank R. Hackbarth and J. Palframan for providing instances of succession and better understanding of the offshoring process.

References

[1] A. J. Albrecht and J. R. Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE Trans. on Software Engineering*, 9(6):638–648, 1983.

[2] D. Atkins, T. Ball, T. Graves, and A. Mockus. Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering*, 28(7):625–637, July 2002.

[3] V. Basili and R. Reiter. An investigation of human factors in software development. *IEEE Computer*, 12(12):21–38, December 1979.

[4] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[5] L. Briand, P. Devanbu, and W. Melo. An investigation into coupling measures for c++. In *Proceedings of the 19th international conference on Software engineering*, pages 412–421, Boston, MA, 1997.

[6] M. Cataldo, P. Wagstrom, J. Herbsleb, and K. Carley. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *Conference on Computer Supported Cooperative Work CSCW'06*, Banff, Alberta, Canada, 2006.

[7] B. Curtis. Substantiating programmer variability. In *Proceedings of the IEEE 69*, July 1981.

[8] J. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *2003 International Conference on Foundations of Software Engineering*, Helsinki, Finland, October 2003. ACM Press.

[9] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*, pages 492–501, Shanghai, China, 2006.

[10] A. Mockus. Large-scale code reuse in open source software. In *ICSE'07 Intl. Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, Minnesota, May 21 2007.

[11] A. Mockus. Software support tools and experimental work. In V. Basili and et al, editors, *Empirical Software Engineering Issues: Critical Assessments and Future Directions*, volume LNCS 4336, pages 91–99. Springer, 2007.

[12] A. Mockus and J. Herbsleb. Expertise browser: A quantitative approach to identifying expertise. In *2002 International Conference on Software Engineering*, pages 503–512, Orlando, Florida, May 19-25 2002. ACM Press.

[13] A. Mockus and D. Weiss. Interval quality: Relating customer-perceived quality to process quality. In *2008 International Conference on Software Engineering*, pages 733–740, Leipzig, Germany, May 10–18 2008. ACM Press.

[14] A. Mockus and D. M. Weiss. Globalization by chunking: a quantitative approach. *IEEE Software*, 18(2):30–37, March 2001.

[15] M. P. Robillard and W. C. G. C. Murphy. How effective developers investigate source code: An exploratory study. *tse*, 30(12):889–903, 2004.

[16] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantir: Raising awareness among configuration management workspaces. In *25th International Conference on Software Engineering (ICSE'03)*, page 444, 2003.

[17] E. Schein. The individual, the organization, and the carier: A conceptual scheme. *Journal of Applied Behavioural Science*, 7:401–426, 1971.

[18] J. Singer, R. Elves, and M.-A. Storey. Navtracks: Supporting navigation in software. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 173–175, 2005.

[19] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby. Shared waypoints and social tagging to support collaboration in software development. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, Banff, Alberta, Canada, 2006.

[20] J. Van Maanen and E. Schein. Towards a theory of organizational socialization. In B. Staw, editor, *Research in organizational behavior*, volume 1, pages 209–264. JAI Press, Greenwich, CT, 1979.

[21] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, July 2003.