Transfer of Code Ownership, Implicit Teams, and Organizational Tomography

Audris Mockus

March 13, 2008

Abstract

Transfer of code ownership or *succession* is a crucial ingredient in open source projects because the source code tends to be reused and in global software development because products are offshored or outsourced. Measuring and studying such transfer can highlight the way organizations adapt to new product structure and change the received product in response. We evaluate several measures of succession on a sample of developers whose tasks were transferred. The best fitting measures were then used to discover the most likely relationships for multiple development locations. To illustrate some of the powerful implications we propose *productivity ratio* to measure the decrease in productivity associated with the succession and quantify it for instances of within- and across-location succession. Decrease in productivity ratio almost doubles when comparing across-location to within-location successions.

1 Introduction

Present software development business practices are trying to emulate the successes of manufacturing process by offshoring software development to countries with lower labor costs. 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 some possible reasons for that challenge. We look at a very specific aspect of code ownership transfer, where the offshore developers take ownership of the code that was initially created by developers in another country. Our analysis primarily focuses on ramifications of code ownership transfer in general. However, we investigate this problem in the offshoring context, therefore some aspects of the findings are likely to be specific to that context.

At the conceptual level, the code ownership transfer, among other aspects, leads to a change in organizational structure without the corresponding change in product structure. Therefore, we expect to see some changes in product structure (or a change in the receiving organization structure) as a result of such transfers. Finding such organizational and product structure evolution may provide insights and recommendations related to efforts directed to code ownership transfer in particular and for improvements of organizational and technical structure in general.

Given the difficulty of measuring the concepts involved, we are focused on reframing the concepts to reflect the same or similar phenomena *and* be subject to measurement, and on techniques that demonstrate various aspect of changes in 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 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 code ownership 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].

Our primary objective is to come up with measures of implicit teams and of the succession in them. Because observations available to us involve only projections of the organizational structure on the work product and IT support systems, the attempt to reconstruct organizational structure is analogous to image tomography (image, often three-dimensional, reconstruction from multiple projections), thus we use the term organizational tomography to describe methods reconstructing organizational structure from the projections or traces in version control, problem tracking, and other software support and IT systems.

The ultimate objective of this work is to relate implicit team structure and succession to aspects crucial to software engineering: cost, quality, and speed. However, we need to develop reliable measures of organizational dynamics reflected through succession and other measurable phenomena in order to achieve this goal.

We start from an empirical study where we use known instances of succession to identify observable patterns in software change and problem reporting data. We then apply these patterns to determine succession in other projects and parts of the product and propose a method to validate these results. Finally, we investigate if the relevant software engineering outcomes change as a result of the succession.

2 Context

We investigate a medium sized (1-3 MNCSL) project that has engaged in offshoring practices for several years and has built a substantial expertise in offshoring. In broad strokes, the practice identifies the tasks and individuals whose work is a candidate for outsourcing and obtains their cooperation by assigning them different responsibilities or

[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.

by providing a separation bonus contingent on expertise transfer. At the same time, a small team of developers in the outsourcing location is identified and their team lead is brought to the outsourcee location to follow (shadow) the outsourcee's work by participating in all meetings, phone calls, and other business related activities together with the outsourcee. After a few weeks of shadowing, the team lead returns to the outsourcing location and trains remaining members of the team who take over the duties of the outsourcee. 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 outsourcee around, and the aspect of learning: one that follows the opinions or teachings of another.

In particular, we have identified 14 individuals and their followers to use as our training set to detect patterns of code ownership transfer. Four of these individuals were not involved in development tasks, therefore we had only ten pairs representing succession.

3 Measurement

Our measurement framework consists of two parts: the version control system logs and the developer and manager interviews. The source of training data and the qualitative description of the outsourcing process were obtained from the interviews and casual conversations, while the code ownership measures were obtained from the version control system.

Our first task was to infer code ownership transfer patterns based on the data obtained from interviews. To accomplish this we postulated five different measures of succession and ranked all 3.5K present and former company developers based on how close they were to each outsourcee.

The measures were constructed based on the idea that modifying the same set of files indicates shared or transfered code ownership. Thus, these measures are also indicators of implicit teams as well.

Denote files as $f_i,\ i = 1, \ldots, N$, developers $d_j, j = 1, \ldots, M$, and the time of changes $c_k(f_i, d_j), k = 1, \ldots, K_{ij}$. The first measure S_0 counts files where the first (in the temporal sense) 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}:

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

The idea behind the first measure is to capture the the temporal aspect in change of ownership or succession when one developer changes the file after another developer. This measure treats all files equally, however some files may be more relevant to the succession.

The second measure S_1 also takes the temporal aspects into account and weights each file by the fractions of changes developers made on that file. Denote the number of changes developer j made to file i as n_{ij}, then:

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

This way the files central to each developer get more weight and have 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 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 would contribute a lot.

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

S_2 also ranges from zero to two, with 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:

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

The fifth measure S_4 is derived from the third measure by calculating the amount of asymmetry in opposite directions:

$$S_4(d_{j_0}, d_{j_1}) = \frac{S_2(d_{j_0}, d_{j_1})}{S_2(d_{j_1}, d_{j_0}) + \epsilon}$$

The idea is that developers with most asymmetry would be the ones that were well separated temporally (one of them starting to modify files later) over all the files they worked on and, therefore, representing good candidates for the transfer of ownership. 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.

4 Evaluation of succession measures

To evaluate these five measures we have calculated their values for all pairs of the ten followers and all the remaining developers. For each follower we thus got a list of values for each measure that was sorted by magnitude in the decreasing order. That way each follower got a list of all potential outsourcees ordered by a particular measure.

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

Looking at a particular follower and a particular measure the first developer in this ordered list represents the best candidate (according to that measure) for being the outsourcee for that follower according (or being on the same implicit team). We then looked at the rank (position in this ordered list) of the actual outsourcee. These ranks (starting from zero) are presented in Table 1.

Follower	S_0	S_1	S_2	S_3	S_4
1	2	0	1	1	1
2	20	51	23	25	14
3	11	9	20	7	14
4	56	126	19	111	12
5	0	5	2	4	0
6	9	44	5	4	4
7	10	81	3	39	2
8	0	9	0	0	25
9	8	35	9	13	17
10	2	39	0	0	0
Totals	118	399	82	204	89

Table 1: The ranks of interview-derived outsourcees according to five measures for 10 followers.

There are several patterns visible in the table. First, the measure S_1 does not appear to provide good results despite its intuitive appeal and the measure S_2 appears to be uniformly better than the rest. 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. This is true to the extent that the measure S_3 incorporating both weights appears to be inferior to measure S_2 that looks only at one weight.

Surprisingly, purely temporal first measure S_0 appears to perform quite well in detecting the outsourcee-follower relationships. It also appears that by combining measure S_2 with the extent of its asymmetry (S_4) may improve the detection of succession.

Second observation concerns several outsourcees (2, 3, 4, and 9) that have followers that go on to do work on parts of the code that appear not to be closely related to what the outsourcee was changing because there are many better candidates according to all measures. A closer look at these instances reveals other developers that had ownership of the code prior to the follower taking it over, but after the outsourcee started working on it. 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 at communication traces among developers. It is nearly impossible to obtain such communication paths 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 score for the ranks that could be achieved is zero, yet we get 82 as a sum of all scores for the measure S_2 (the sum is only 11 if we exclude followers 2, 3, 4, and 9). 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 the best we can expect, while the scores for the remaining followers may be improved through better measures that take into account entire succession graph as suggested above. On the positive side, the worst possible score would be 35K if the true outsourcee was the last of the 3.5K developers for each follower. A more reasonable worst-case score would be having the true outsourcee to be the last among developers that changed at least one file in common. This worst possible measure for our sample of followers is 1033 and the range varied from 113 to 153 potential (that changed at least one file in common) outsourcees among the ten followers.

5 Inferring succession

Based on the experiences of fitting outsourcee-follower relationships described in Section 4, we applied the best performing measure S_2 on a sample of 134 potential followers to find their most likely outsourcees. The potential followers were selected by choosing developers with at least 100 changes that were located at the country (according to the country code of their telephone number) where most of the outsourcing is directed to. This set of developers was no longer restricted to the project on which the measures were evaluated in the previous section.

There were 84 distinct outsourcees identified using the top outsourcee candidate via measure S_2. The number of outsourcees is lower than the number of followers because a team of followers tends to take over tasks assigned to one outsourcee. The outsourcee with the largest number of followers had 11 followers, three outsourcees had five followers, 20 had between two and four, and 60 outsourcees had one follower.

It would be of interest to conduct a study to verify how many of these relationships were completely explicit, i.e., the follower was assigned to shadow the outsourcee, and how many were completely implicit, i.e., where the follower may not be aware of the outsourcees existence. This can be accomplished by interviewing a sample of the developers involved in succession.

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. To illustrate a narrow aspect of the relevance of the succession phenomena we compared the productivity of outsourcees and their followers in the sample selected in Section 5.

To estimate productivity we calculated the average number of changes (delta) per month each developer made. The geometric average[3] over outsourcees (counting each of the 84 outsourcees only once) was 87 delta per month while the average for the followers was 27 delta per month. Even if we exclude any outsourcee with more than one follower, we get the geometric average of 68 delta per month over the sixty outsourcees with a single follower.

The difference in productivity is in line with the practice of assigning teams of developers for each outsourcee and roughly quantifies

[3]The distribution of productivity is highly skewed, therefore geometric average is more sensible than arithmetic average.

the productivity relationship in the succession context. It would be of interest to verify if such productivity ratios are specific to outsourcee-follower relationships or hold for more general scenarios of succession. It should be noted that the productivity ratio does not compare the productivity of developers in various locations. To do that we would need to select developers of comparable experience on both sides, an unlikely scenario in an outsourcee-follower relationship. Furthermore, it is likely that the productivity ratio depends on the type of succession. Much of the code transfer involves deployed releases that need current engineering support (fix customer reported problems and deliver patches). Productivity of fixing customer reported defects (in terms of delta per month) tends to be lower than for the development of a new release. Therefore, we hypothesize higher productivity ratios in succession instances where new feature development is offshored.

To test some aspects of this hypothesis we have conducted a further analysis. We inferred succession as described in Section 5 for all locations and investigated the productivity ratio for the instances of succession within a single location and across locations shown in Table 2. The within-location transfers are most likely to include new feature development, while cross-location transfers are more likely to involve current engineering work.

From	To	Productivity ratio	Number of pairs
A	A	0.41	76
B	B	0.43	32
C	C	0.24	32
A	B	0.29	17
C	B	0.11	22
A	C	0.19	76

Table 2: The productivity ratio for the succession across and within locations (there are fewer than 10 developer pairs for the remaining location combinations).

Table 2 shows that locations A and B have a similar and relatively high within-location productivity ratio and location C has a substantially lower ratio. In all cases, across-location succession substantially lowers the the productivity ratio from that of within-location succession. This suggests that a mismatch between organizational structure of outsourcee and follower organizations may have an impact on the productivity reduction. Other factors besides the type of work that was transferred are also likely to affect the productivity ratio and their relative contributions need to be investigated. For example, the structure of the product, the structure of organizations on the giving and receiving ends of the code transfer, the differences between these organizations, and experience of developers are some of the factors that need to be explored.

7 Conclusions and related work

While the code ownership has been measured before (see, for example, [3, 5]), we have demonstrated the feasibility of measuring a phenomena of succession and assessing how it affects one of key software outcomes. The succession becomes more important as the software development increasingly follows in the offshoring and outsourcing footsteps of the manufacturing and because of code reuse associated with open source software [4]. More generally, the succession is a key aspect of organizational dynamics in software projects.

Organizational tomography has a substantial history. Arguably, the reconstruction of organizational patterns from interviews conducted in [1] represents an instance of organizational tomography. The history of the software product [2] provided some insights about the organization from these projections in change management data and, therefore, can be considered to be an example of organizational tomography. In [6], the code was chunked into independently changeable pieces of suitable size to fit the capabilities of the less expensive offshore development location.

More recently, various aspects of organizational tomography have blossomed through applications in studies of open source projects and global software development that started having regular international research meetings a few years ago.

Clearly, the proof of concept presented in this work requires extensive validation and further development, 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 [4] and the ability to quantify how the product and organization coevolve are likely to provide numerous lessons and 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] Tom Allen. *Managing the flow of technology*. MIT Press, Cambridge, MA, 1977.

[2] S.G. Eick, J.L. Steffen, and Sumner E.E. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957 – 968, November 1992.

[3] A. Mockus, R. F. Fielding, and J. Herbsleb. A case study of open source development: The apache server. In *22nd International Conference on Software Engineering*, pages 263–272, Limerick, Ireland, June 4-11 2000.

[4] Audris 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.

[5] Audris Mockus and James 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.

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