Succession: Measuring Transfer of Code and Developer Productivity

Audris Mockus

audris@avaya.com

Avaya Labs Research

Basking Ridge, NJ 07920

http://mockus.org/

Outline: Measuring and Using Succession

- Motivation: code as functional knowledge
- How to measure succession?
- How succession affects developer productivity?
- Implications

Premise: Code as Functional Knowledge

- Functional knowledge
  - Scholarly and literary works need a subject to interpret/perform them
  - Code just needs a computer to be executed
- Open source code
  - A vehicle for innovation through reuse (build on existing knowledge)
  - A common platform for everyone to express themselves (contribute their knowledge)
- Codebase for legacy systems encodes thousands of person-years of knowledge on:
  - the organization (process of producing the code) and
  - the market (value the software provides to users)

Code as functional knowledge: implications

- Developers are *transient*, but the code is *everlasting*
- Developers can *only* leave a lasting impact
  - through *changes* to the code and
  - through *training* developers who succeed them
- Traces developers leave in the code shed light on how software is created and how developers interact
- **What happens in a succession: when developers change, but code stays?**

What are we doing: domain and method

- Science
  - X is the study of *past human events and activities*
  - Y is the study of human **cultures** through the *recovery, documentation and analysis of* **material** *remains*
  - Z is the study of human **teams** through the *recovery, documentation and analysis of* **digital** *remains*
- Method
  - Tomography is image reconstruction from multiple projections
  - Organizational tomography is the reconstruction of structure and behavior of a team from the digital traces it leaves in the code and elsewhere

Data sources

- People: organizational Directory (LDAP) snapshots
  - Chronology: late 2001 and early 2003. Early 2004 until present: weekly extracts.
  - Attributes: personal ID, supervisor ID, department, location, phone, email
- People to login maps
  - Yellow pages (NIS), weekly extracts from three clusters
    - login to LDAP attributes, name
  - Proprietary problem reporting system (QQ), weekly extracts
    - login to name, email
- Version control systems
  - Chronology: 1990 until present, varies with project
  - Attributes: login, date, file

Validation: An example problem

- Problem: people change logins over time
  - Change in IT infrastructure
  - Change in security/authentication requirements
- Solution: detect such change and map login to the person:
  - Map login to organizational IDs from LDAP (POST/ActiveDirectory)
  - Problem: LDAP IDs also change over time
  - Solution: construct an ID unique to the person
    - Use multiple attributes: for example, if ID changes but name/phone/location do not
      - If IDs do not overlap in time then conclude that ID changed

Defining succession

- *Definition: Implicit* or *virtual teams* are relationships among individuals based on the affinity to the parts of the product they are working or have worked on.
- *Definition: Succession* is a relationship between individuals within the implicit teams reflecting the transfer of responsibilities to maintain and enhance the product. There are various types of succession: here we are concerned with offshoring and refer to receiving party as *followers* and to the transferring party as *mentors*. Note that, in general, followers and mentors do not need to communicate with each other.
  - Other successions types: new developers in an organization, code reusers in OSS and other projects
- Objective: measure *succession* and its impact.

Projections and the inverse problem

- Projections
  - “Engaging” with the code often leads to changing the code (here we do not consider non-developer roles of a tester/documenter)
  - The chronological order of engagements by *mentors* and *followers* should be reflected in the temporal order of changes
- The inverse problem (reconstruction or tomography)
  - Implicit teams: developers changing the same packages, files, methods, or lines
  - Succession: pairs of developers with the most clear succession signature
    - More shared code
    - Stronger chronological sequence

Design of succession signatures

- For a developer a, the *mentor* is
  $b = \arg\max_{b \in \{Developers\}} S(a, b)$
  - $S(a, b) > S(a, c)$ if b is more likely than c to be a mentor for a
- For a pair of developers a, b the succession is reflected by:
  - S_0: the number of **files** (packages, methods, or lines) with an earlier **first** (median or last) change by b than the **first** (median or last) change by a.
  - S_1: the number of files weighted by the proportion of developer's changes on that file.
  - S_2: the number of files weighted by the proportion of file's changes made by that pair of developers.
  - S_3: the number of files weighted by the proportion of developer's changes on that file and by the proportion of file's changes made by that pair of developers.

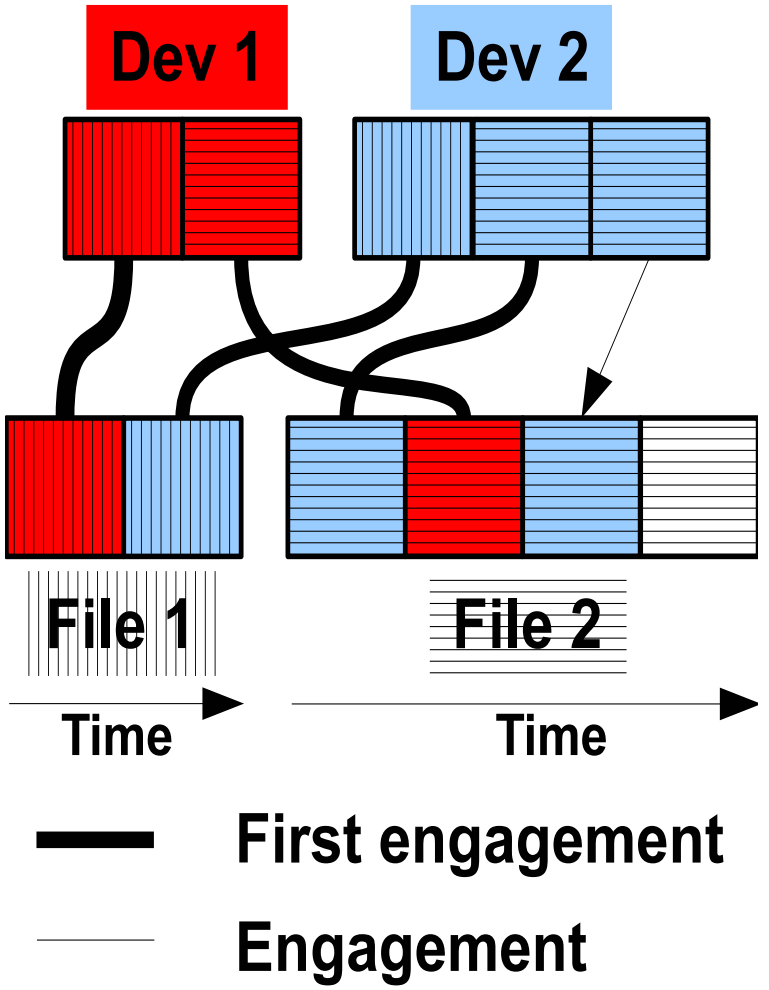
Illustration of succession signatures

$$S_0(d_1, d_2) = S_0(d_2, d_1) = 1 \implies \text{none}$$

$$\begin{cases} S_1(d_1, d_2) = \frac{1}{2} + \frac{2}{3} \\ S_1(d_2, d_1) = \frac{1}{2} + \frac{1}{3} \end{cases} \implies d_1 > d_2$$

$$\begin{cases} S_2(d_1, d_2) = \frac{1}{4} + \frac{2}{4} \\ S_2(d_2, d_1) = \frac{1}{2} + \frac{1}{2} \end{cases} \implies d_2 > d_1$$

$$\begin{cases} S_3(d_1, d_2) = \frac{\frac{1^2}{2} + \frac{2^2}{3}}{4} \\ S_3(d_2, d_1) = \frac{\frac{1^2}{2} + \frac{1^2}{3}}{2} \end{cases} \implies d_1 > d_2$$



Succession and real (not virtual) mentorship

- Pick a follower F_1
- $\forall D_i$ calculate $S_0(F_1, D_i)$ and order: $S_0(F_1, D_0) \leq \cdots \leq S_0(F_1, D_n)$
- Identify real mentor D_k for F_1 via interview
- k is the rank for the real mentor D_k among all developers
  - If S_0 is unrelated to mentorship then k is uniform on $[0, n]$.
  - If S_0 is capturing mentorship: then $k << n$.

Mentor?	i	$S_0(F_1, D_i)$
	0	33
	1	32
$k = 2$	**2**	**29**
	3	11
	4	10
	5	9
	6	7
	...	...
	127	0

Four succession signatures for ten followers

Ten mentor-follower relationships established via interviews

Performance of the signature is based on the rank of interview-identified pair among all other pairs involving the follower

Flwr.	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	Tot.
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
Tot.	4	**119**	**57**	**312**	11	48	110	17	**82**	41	803
p-val S_2	.01	**.15**	**.12**	**.12**	.016	.03	.019	.008	**.05**	.006	.05

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

Analyzing succession: theoretical framework

- Organizational socialization
  - New entrants to an organization learn organizational culture and, based on how successfully they can practice that culture, they move up the hierarchy and to the center of decision making
  - The result of such cultural socialization may be:
    - *Custodial*: the newcomer completely preserves organizational traditions
    - *Innovative*: the newcomer changes information or mission of their organizational function
  - Organizational culture is a result of organizational evolution: organizations that can not preserve it (have too innovative culture) or can not adjust it in response to changing environment (have too custodial culture) disappear

Organizational socialization: succession

- Hypotheses
  - Offshoring succession is less informal/random $\Longrightarrow$ less innovation
  - Mentors with expertise dispersed over several products would provide mentorship that leads to more innovation
  - Mentors that transfer expertise of their secondary products would lead to less innovation by the followers
  - Mentors with more followers would have less innovative followers
  - Products with the oldest and largest code bases are likely to have lower productivity ratios
  - The effectiveness of expertise transfer increases over time as the organization improves its offshoring practices
- Custodial responses are likely to lead to a lower productivity ratio because followers will have to learn from mentor's example
- Productivity: number of delta (atomic changes) per month

The productivity ratio model

Table 2: $\log(Ratio) = Time + Offshore + Primary + Breadth + Size + \log(NFollow)$. 1012 mentor-follower pairs. Adj-$R^2 = 59$.

	Estimate	p-value	e^{est}	95%CI
Time of transfer	-0.01	0.31		
Offshoring	-0.63	0.00	$\frac{1}{2}$	$[0.42, 0.67]$
Primary expertise	-0.68	0.00	$\frac{1}{2}$	$[0.40, 0.64]$
Expertise breadth	-1.41	0.00	$\frac{1}{2}$	$[0.38, 0.64]$
Large prod.	-1.21	0.00	$\frac{1}{3}$	$[0.21, 0.42]$
Medium prod.	-0.46	0.00	$\frac{2}{3}$	$[0.52, 0.77]$
$\ln(NF)$	-0.53	0.00	$\sqrt{NF}$	

Organizational Tomography and Succession

- Succession phenomena: code stays, developers come and go
  - Estimable: mentor-follower relationships are traceable
  - Succession has steep costs/practical implications
    - offshoring: need two to replace one
    - size: need three for largest products, one for smallest
    - expertise of the mentor
      - transferring secondary expertise: two for one
      - narrow mentor's expertise: up to two for one
    - too many followers: square root of the number of followers per mentor
- Tomography: inverse problem approach adds rigor to empirical studies based on project data

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. Using 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 formulated using 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.

Audris Mockus
Avaya Labs Research
233 Mt. Airy Road
Basking Ridge, NJ 07920
ph: +1 908 696 5608, fax:+1 908 696 5402
http://mockus.org, mailto:audris@mockus.org

Audris Mockus is interested in quantifying, modeling, and improving software development. He designs data mining methods to summarize and augment software change data, interactive visualization techniques to inspect, present, and control the development process, and statistical models and optimization techniques to understand the relationships among people, organizations, and characteristics of a software product. Audris Mockus received B.S. and M.S. in Applied Mathematics from Moscow Institute of Physics and Technology in 1988. In 1991 he received M.S. and in 1994 he received Ph.D. in Statistics from Carnegie Mellon University. He works in the Software Technology Research Department of Avaya Labs. Previously he worked in the Software Production Research Department of Bell Labs.