Transfer of Code Ownership, Implicit Teams, and Organizational Tomography [1]

Audris Mockus

audris@avaya.com

Avaya Labs Research

Basking Ridge, NJ 07920

http://mockus.org/

Code as functional knowledge

- Premise
  - Code reuse is good, e.g., “modules reused without revision had the fewest faults, fewest faults per source line, and lowest fault correction effort” [2]
  - Codebase defines the organization and the market for some legacy systems
  - Open source code
    - A vehicle for innovation through reuse
    - A common platform for multiple participants
  - Offshoring
    - Rapid transfer of code ownership
- Conclusions
  - Developers are *transient*, but the code is *everlasting*
  - It make sense to study the teams via their traces in the code

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

Methodology

- Select phenomena for the study.
- Observe and validate on a smaller scale how the phenomena gets projected onto digital artefacts.
- Design and validate models of the projection based on the previous step.
- Choose a suitable penalty/regularization/likelihood function for the inverse projection/estimation based on the models and validate based on the observations.
- Infer phenomena and their impacts via inverse projection/estimation on the entire population of interest.

Example phenomenon: 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: we are concerned with offshoring and refer to receiving party as *followers* and to the transferring party as *mentors*.
  - 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 (regularizing or penalty function, entropy or likelihood)

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:
  - H_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.
  - H_1: the number of files weighted by the proportion of developer's changes on that file.
  - H_2: the number of files weighted by the proportion of file's changes made by that pair of developers.
  - H_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.

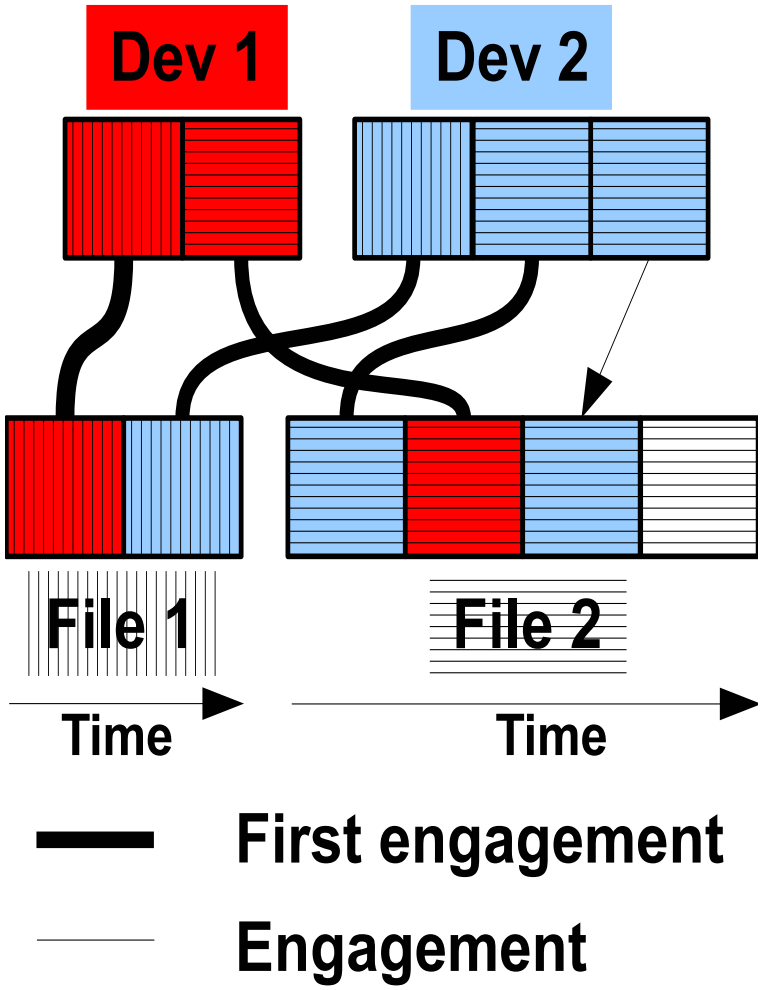
Illustration of succession signatures

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

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

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

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



Evaluation of succession signatures

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

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

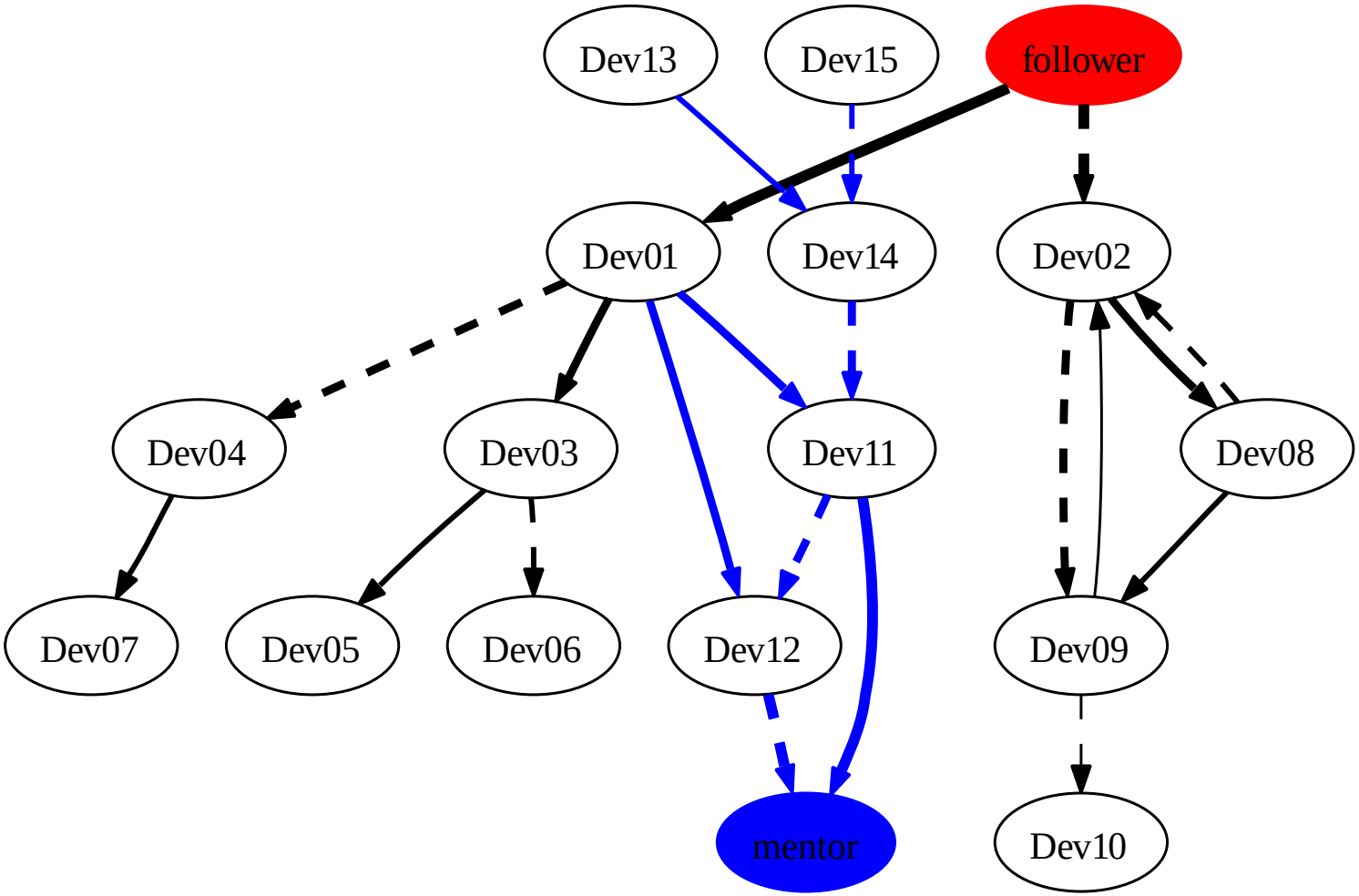
Table 1: The ranks (starting from 0) of interview-derived mentors according to five measures for 10 followers.

Interpretation of signature performance

- The simplest signature S_0 works surprisingly well.
- The most intuitive S_1 (weighting by the fraction of developer's changes on a file) is worst.
- Weighting by the fraction of file's changes made by a developer (S_2) works best.
- Trainees 2, 3, 4, and 9 were off by all signatures: the mentors for them was also mentors for many other developers that were closer to 2, 3, 4, and 9.
- Best total of 0 is not realistic given multiple trainees per mentor. For comparison, worst possible score is 1033, average for a random selection is around 500.

Outliers: Follower 2

- Use the top two values of S_2 to a depth of 3



The ratio of trainee and mentor productivity

- Infer trainee-mentor relationship using S_2
- Productivity: number of delta (atomic changes) per month

From	To	Geometric Mean	Std. Absolute Error	Number of pairs
A	A	0.41	0.05	76
B	B	0.43	0.08	32
A	B	0.29	0.06	17
C	C	0.24	0.05	32
C	B	0.11	0.01	22
A	C	0.19	0.07	6

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

Lessons: Tomography and Succession

- Tomography
  - Analyzing software phenomena from code traces using an inverse problem approach is a feasible way to add rigor to empirical studies of software
- Succession phenomena
  - The code ownership dynamics appears to be:
    - Estimable: mentors can be traced
    - Relevant: affects productivity

Team dynamics and other phenomena

- How are other software phenomena projected onto digital artefacts?
- What are the best reconstruction models and methods for the organizational tomography?
- Investigate succession
  - Validate the signature, validate the inferred phenomena (succession), validate on the universe of all SW projects (OSS)
  - Recognize different types of succession (e.g., outsourcee vs mentor)
  - Other types of impact: quality, lead-time

References

[1] Audris Mockus. Transfer of code ownership, implicit teams, and organizational tomography. Technical Report ALR-2008-010, CID 135147, Avaya Labs research, 2008.

[2] Richard W. Selby. Enabling reuse-based software development of large-scale systems. *IEEE Trans. on Software Engineering*, 31(6):495–510, June 2005.

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.

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.