Developer Fluency: Achieving True Mastery in Software Projects

Minghui Zhou,
zhmh@pku.edu.cn,
Peking University,
Beijing, China

Audris Mockus
audris@avaya.com
Avaya Research Labs,
NJ, USA

Agenda

- History
- Motivation
- Methodology
- Results

History of developers' competence

Claim the issue

Individual differences among project personnel accounts for the largest source of variation in project performance

Sackman et al, 1968, 28:1;
Curtis, 1981,23:1;
Boehm, 1981

Claim the methodology

"By using …source code change history and problem reports we quantify aspects of developer participation, …, productivity..."

Mockus et al, 2000

Time

"The initial attempt had failed poorly…"

"Until the many sources of variation among individuals have been compared in the same set of data, it will not be possible to determine …the most important predictor of success…" -Curtis, 1984

Recent findings

How developers new to the project learn

Von Krogh et al. looked at the strategies and processes by which newcomers join the existing OSS community.. -2003

Dagenais et al listed obstacles facing developers joining projects through observing 18 IBM developers -2010

……

Motivation

Offshoring/outsourcing

- "all (outsourcing) teams have similar experience levels, and all have had an influx of graduates and are struggling to get them up to speed“ – Outsourcing manager

➢ How to speed up the project newcomers?

Organization strategy

➢ Massive retirement of core developers in mature legacy products started in the 90's

- ``Original developers probably understood how features would work and what feature interactions worked, but subsequent developers are not necessarily aware of the whole context'' –Top developer

➢ How should the newcomers learn about the product?

"productive" ≠ "competent"

- How long does it take for a developer in your project to become productive?
  - Small-medium scale projects: 2-6 months
  - Large scale project: 12 months
- What are the stages for a developer?
  - Small-medium scale projects : "it takes *several years* to become competent in important tasks"
  - Large scale project : "we had attempted to assign mentoring tasks to developers with *only two years* of experience, but had unsatisfactory results"

Research question

How long does it take for an average developer to become _fluent_ in a software project?

Fluency: Complete project tasks rapidly and accurately independent of task difficulty or importance.

Methodology

Qualitative approach

- Clarifying the purpose,
- Designing questions and subjects,
- Interviewing and transcribing,
- Analyzing,
- Validating/verifying, and
- Reporting

Quantitative study

- Retrieve the raw data,
- Perform initial cleaning and processing,
- Create measures to answer our research questions, perform analysis of these measures, and
- Validate the results

Proj-ects	Years	Domain	Sites	# of Par-ticipants	Participant role:location
A	> 15	Call center	US offshored to India	4	3 dvlprs:India, DM:India
B	➢10	Dialer	US offshored to India	4	3 dvlprs: India, DM:India
C	> 10	Voice Response	US offshored to India	4	3 dvlprs:India, DM:India
D	> 15	Core telephony	US partly offshored to India	6	3 dvlprs: US, DM: US, OM: US, QM: US
E	➢10	Embedded telephony: endpoints	US offshored to India	2	DM: India, OM: India
F	> 7	Embedded core telephony	UK partly offshored to India and Romania	3	DM: UK, OM: Romania, QM: UK
G	> 15	Messaging	UK and US partly offshored to India	2	DM: UK, OM: UK
H	>5	Contact Center	US partly offshored to India	2	DM: US, OM: US
I	3	Middleware	China	4	3 dvlpers: China, DM: China
J	2	A web-based development platform	China	4	3 dvlprs: China, DM: China

Data

- Raw data
  - Code changes from version control systems including cvs, svn, clearcase, sccs
  - MRs from issue tracking systems including Jira, Sablime, propriatary system
- Observations
  - 20544 changes, 85 developers in Project D
  - 13081 changes, 69 developers in Project A,B and C

Results

"productive" ≠ "competent"

- How long does it take for a developer in your project to become productive?
  - A-J(except D): 2-6 months
  - D: 12 months
- What are the stages for a developer?
  - A-J(except D): "it takes *several years* to become competent in important tasks"
  - D: "we had attempted to assign mentoring tasks to developers with *only two years* of experience, but had unsatisfactory results"

Why fully productive developers are not assigned some important project tasks?

Task variations

Interview questions

- What tasks did you do when you joined the project? What was your project? Which part of the project did you work on: e.g., developing a new feature, fixing bugs (current engineering)?
- What tasks are you doing now? …

Task variations have two sides

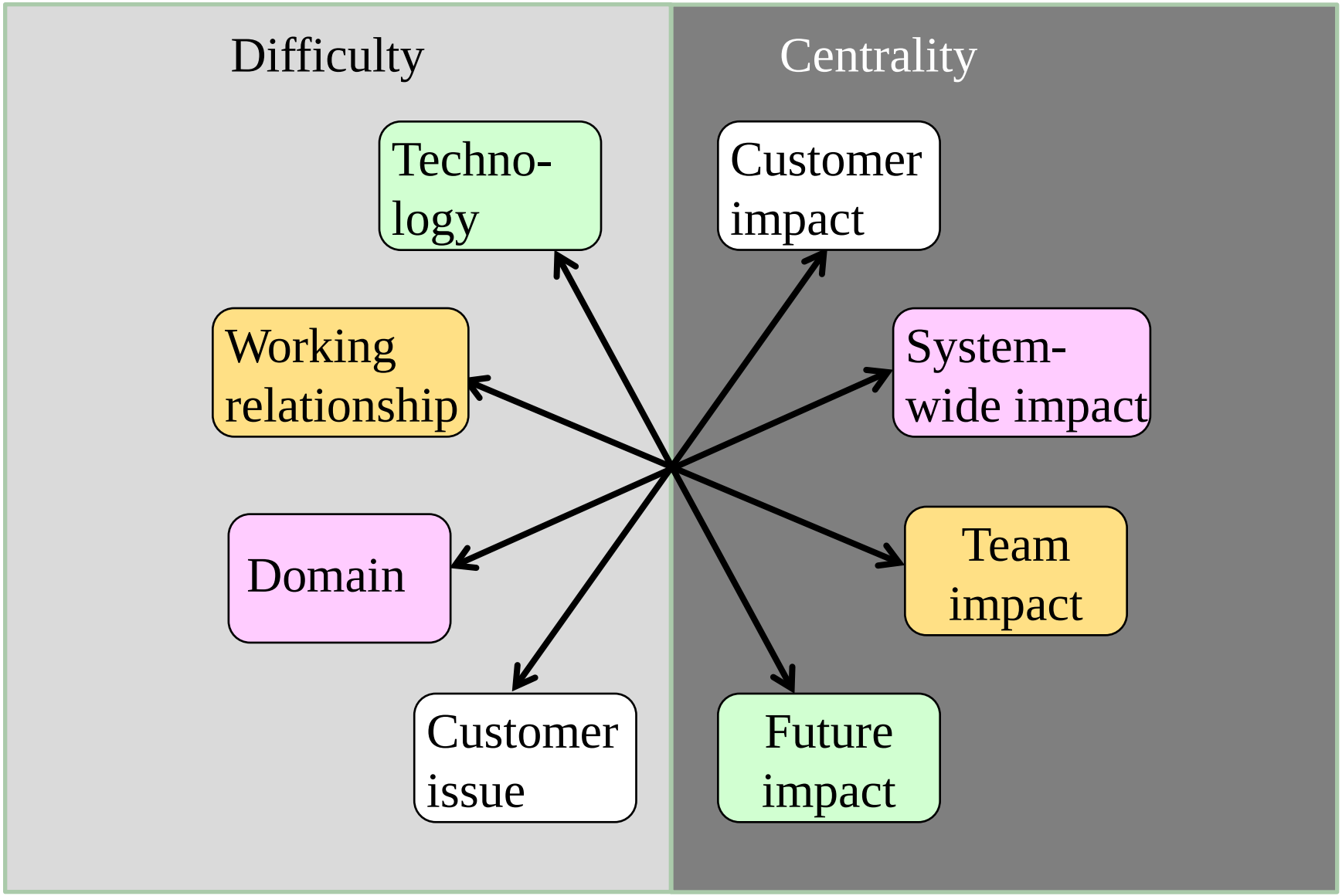
Difficulty is not centrality

- "the effort to complete an MR is not a factor in assessing the importance of the MR" –manager of G

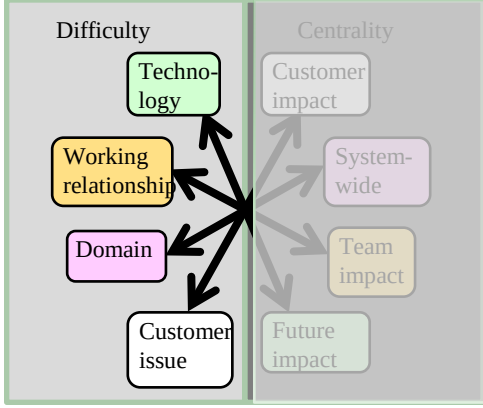
Difficulty overlaps with centrality

- "it's always easier to do something that doesn't involve lots of people" –tester of D

Task difficulty and task centrality

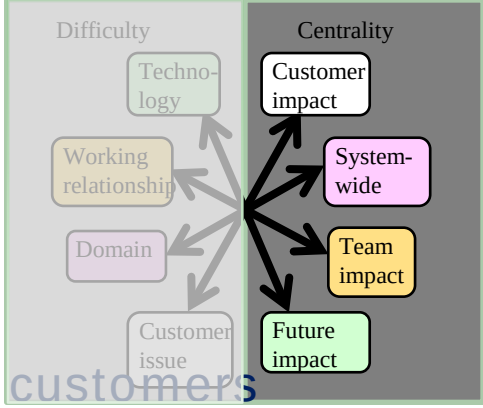


Task difficulty



- Technology,
  - "Java is easier than C++" - developers from I
- Domain . In a product, some domains are considered to be more difficult than others.
  - "this forge module is a mess, it has too many relationships with other modules." -developer from J
- Working relationships. A task which requires communications with more people is considered to be more complicated
  - "it's always easier to do something that doesn't involve lots of people" –tester from D
- Customer related issues.
  - "A developer found defect is always simpler to fix than a bug found by customers." - manager from G

Task centrality



- Customer impact
  - D, “customer escalation trumps everything”;
  - I, “the most experienced developers are sent to the customers to resolve their problems.”
- System-wide impact
  - J, “there are two most important modules, one is the common library, all the other modules would invoke them; the other is the forge module, which needs to invoke all the other modules and show them to the users.”
- Team impact
  - J, “once I found some developer who didn’t write comments in their committing changes, I would go to them and ask them to add them and do that in the future.”

.

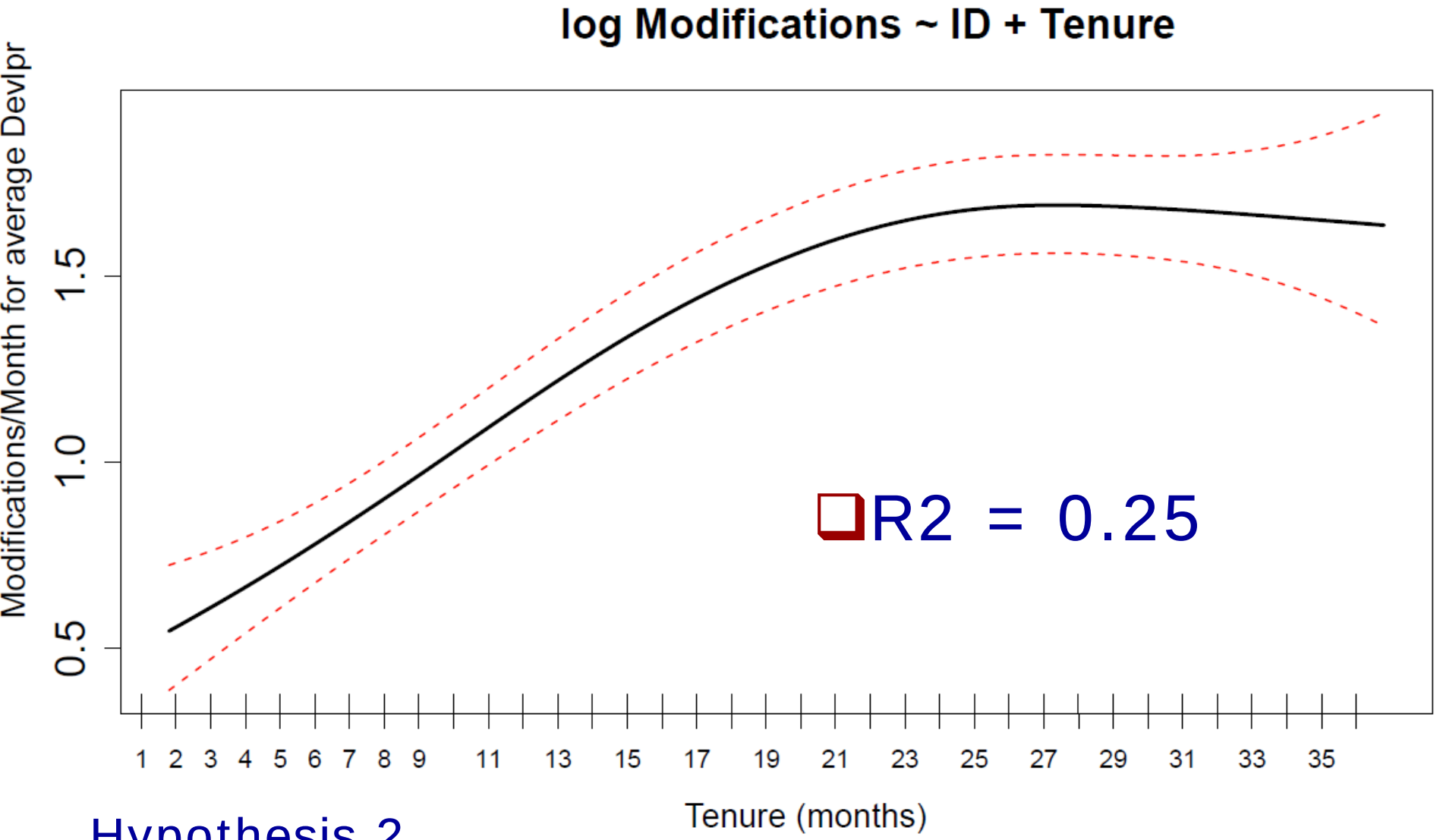
- Future impact
  - D, “I see a sense of urgency for our team in terms of skill acquisition so the team is equipped to address the next generation of software and product technologies.”

Hypothesis 1.

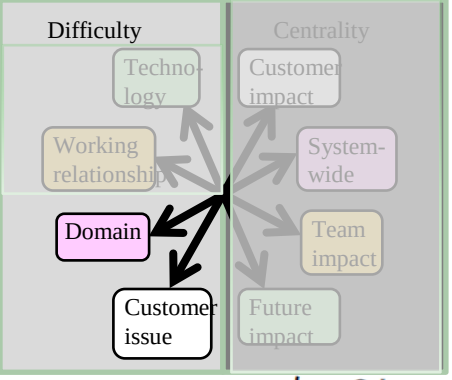
In a software project tasks vary in terms of difficulty and centrality. Different tasks require different degrees of project fluency.

Quantify how a developer's fluency grows over time

- Number of tasks (modifications) per staff-month
- Productivity adjusted for task difficulty
- Task centrality

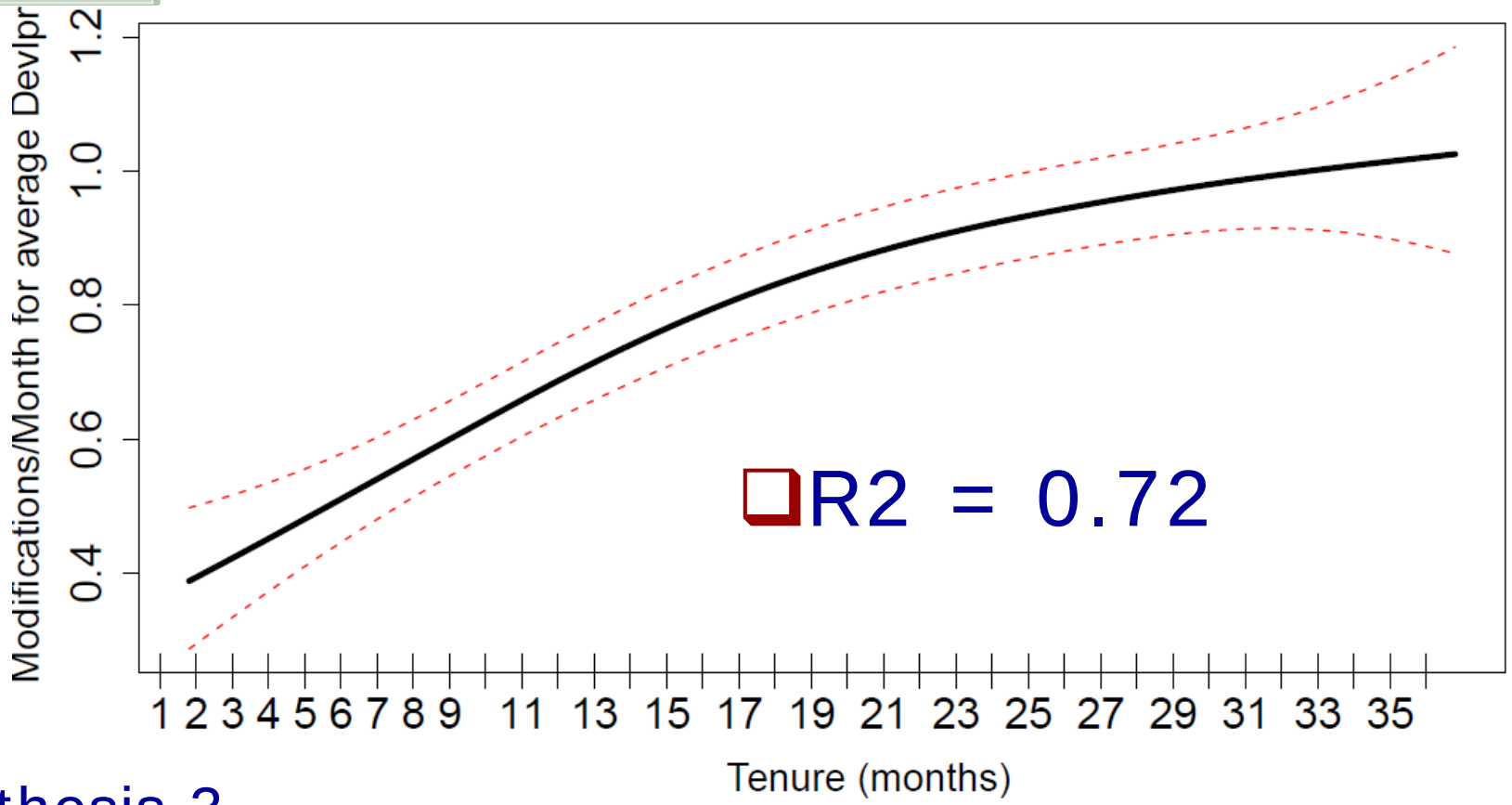


Hypothesis 2.
Developers' productivity plateaus within 6-7 months in small and medium projects and it takes more than 12 months in large projects.

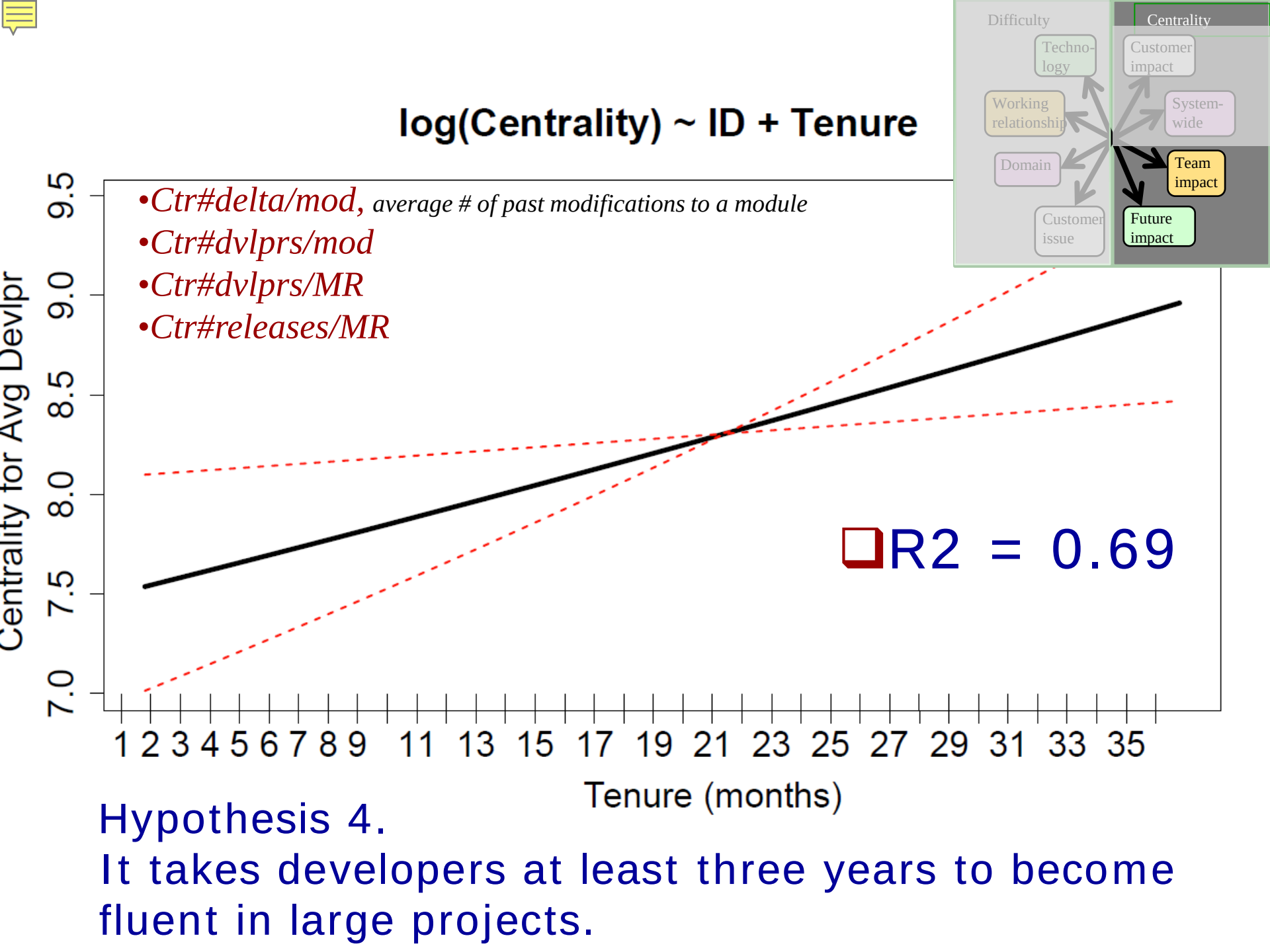


- *AvgFiles:* *The average # of files modified by a task*
- *FractionCust:* *The percentage of tasks related to customer reported issues*

log Modifications ~ ID + AvgFiles + FractionCust + Tenure



Hypothesis 3.
Developers take longer to reach full productivity if we adjust for the difficulty of tasks.



Hypothesis 4.

It takes developers at least three years to become fluent in large projects.

Conclusion

Main findings

- Separate the tasks into four dimensions of difficulty, and four dimensions of centrality,
- Propose ways to measure them, and
- Quantify the growth of a developer's fluency.

Practical implications

- The offshoring schedule has to accommodate longer training periods.
- It may require retaining some existing experienced staff.

Reference

- A. Begel and B. Simon. Novice software developers, all over again. In International Computing Education Research Workshop, Sydney, Australia., 2008.
- C. Binder, E. Haughton, and B. Bateman. Fluency: Achieving true mastery in the learning process. Technical report, University of Virginia Curry School of Special Education, 2002. Professional Papers in Special Education.
- B. Curtis. Substantiating programmer variability. In Proceedings of the IEEE 69, July 1981.
- B. Curtis. Fifteen years of psychology in software engineering: Individual differences & cognitive science. In ICSE'84, pages 97–106, 1984.
- B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vrie. Moving into a NewSoftware Project Landscape. In ICSE2010, Cape Town, South Africa, May 1-8, 2010.
- T. Graves and A. Mockus. Identifying productivity drivers by modeling work units using partial data. Technometrics, 43(2):168–179, May 2001.
- J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally-distributed software development. IEEE Transactions on Software Engineering, 29(6):481–494, June 2003.
- S. Kvale and S. Brinkman. InterViews: Learning the craft of qualitative research interviewing (2nd Ed.). Thousand Oaks, CA: Sage Publications, CA, USA, 2007.
- Lave and E. Wenger. Situated Learning. Legitimate Peripheral Participation. Cambridge University Press, Cambridge, 1991.
- V. J. Marsick and K. E. Watkins. Informal and incidental learning. New Directions for Adult and Continuing Education, 89:25–34, 2001.
- 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.

Reference

- A. Mockus. Succession: Measuring transfer of code and developer productivity. In 2009 International Conference on Software Engineering, Vancouver, CA, May 12–22 2009. ACM Press.
- 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.
- A. Mockus and D. M. Weiss. Globalization by chunking: a quantitative approach. IEEE Software, 18(2):30–37, March 2001.
- F. E. Ritter and L. J. Schooler. International Encyclopedia of the Social and Behavioral Sciences, chapter The learning curve, pages 8602–8605. Pergamon, Amsterdam, 2002.
- P. Robillard. The role of knowledge in software development. Communications of the ACM, 42(1):87–92, 1999.
- S. E. Sim and R. C. Holt. The ramp-up problem in software projects: A case study of how software immigrants naturalize. In ICSE 1998, pages 361–370, 1998.
- 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.
- 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.
- S. Wood. Fast stable direct fitting and smoothness selection for generalized additive models. J.R.Statist.Soc.B, 70(3):495–518, 2008.
- Y. Ye and K. Kishida. Toward an understanding of the motivation open source software developers. In ICSE 2003, pages 419–429, Portland, Oregon, 2003.
- M. Zhou, A. Mockus, and D. Weiss. Learning in offshored and legacy software projects: How product structure shapes organization. In ICSE Workshop on