Growth of Newcomer Competence: Challenges of Globalization

Minghui Zhou
Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, China
zhmh@pku.edu.cn

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

ABSTRACT

The transfer of entire projects to offshore locations, the aging and renewal of core developers in legacy products, the recruiting in fast growing Internet companies, and the participation in open source projects, present similar challenges of rapidly increasing newcomer competence in software projects. In particular, culture differences, communication complexity, and the rapid influx of developers with little or no project knowledge common in these phenomena pose practical and research questions for software engineering. For example, how do different cultures impact project learning? Are there best practices for competence-enhancing communication? How to learn from the experiences of top developers to improve the training of newcomers? What resources and tools can be provided to help newcomers learn faster and become more productive? These questions sketch a project-learning-focused agenda needed to address outlined challenges. We propose how emerging measurement methods utilizing rich data in software repositories and the theoretical frameworks based on cognitive and organizational science may be applied to address these challenges and to improve understanding of how humans learn.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*sociology*

General Terms

Performance, Measurement, Human Factors

Keywords

Project competence, culture difference, communication, learning trajectory, recommending systems, universal repositories

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

1. INTRODUCTION

The transfer of existing projects to offshore locations raises the question of how to speed up the learning of project's newcomers: "*all (outsourcing) teams have similar experience levels, and all have had an influx of graduates and are struggling to get them up to speed*" in the words of an outsourcing manager.

Similarly, the retirement of core developers in mature legacy products that started in the 90's and earlier, leaves these projects without original creators and raises the question of how the newcomers should learn about the product. A top architect in a 30-year-old call-center project expressed his anxiety: "*We need to make sure people understand the skill of our dwindling set of experts. Original developers probably understood how features would work and what feature interactions worked, but subsequent developers who are adding to the code are not necessarily aware of the whole context.*" At the same time, the boom in Internet applications created many successful Internet companies, whose fast growth demands continuous learning from their employees. "*Most of our projects only last for several months, that really requires the learners to be fast enough*" according to the chief architect of Taobao (top e-commerce site in China).

Furthermore, the explosion of Open Source Software (OSS) development, raises the question of how individuals and companies should learn to participate in the OSS projects. For example, the first author has led a group from Peking University that over the last decade has built and later merged an application server with JOnAS (another open source AS) [1]. Numerous issues had to be recognized and addressed, including culture, language, time-zone, network bandwidth, and, most importantly, accessibility to the knowledge about the code and the development practices.

As Fischer [13] noted, phenomena such as globalization, increasing trends to outsource high-level cognitive tasks, and the need to participate effectively in addressing complex world problems are changing how we think, learn, work, and collaborate. These phenomena pose a crucial challenge: bringing the project newcomers up-to-speed and developing project competence. Thus we need a better understanding of how developers learn in a software project, how to make projects more self-documenting, and how to create innovative resources and tools that can facilitate newcomer learning and productivity.

Human learning has been studied in cognitive science, de-

[1] http://jonas.ow2.org

velopmental psychology, and organizational science. More specifically, the mind and brain, the processes of thinking and learning, the neural processes that occur during thought and learning, and the development of competence are intensively studied. What is perhaps currently most striking is the variety of research approaches and techniques that have been developed and ways in which evidence from many different branches of science are beginning to converge [4]. Therefore, not only can we borrow the insights from these disciplines to understand the developers' learning, for example, as Curtis [9] argued, cognitive science is a paradigm that offers the best opportunity to study and gain control over the largest source of influence on project performance, but also to help understand how humans learn in general.

On the other side, many of these advances have been enabled by better data and better analysis tools. The most salient aspect to the study of developer project competencies [2] is the rich data sources in software project repositories ranging from aggregate measures of release quality to individual actions of a developer recorded by IDEs[3]. Many methods and tools to analyze software data are being developed in the software engineering community, but even more work is outside. It is enabled by increasingly detailed collection of data about users and consumers, and driven by advertisement, retail and other segments of the economy that dwarf software engineering in size. Clearly, methods and tools from these fields will find use in quantifying and solving the individual competence problems, for example, the use of social network measures to show the most influential developers, the summaries of how developers communicate while resolving problems, and the studies of culture differences.

2. LEARNING IN GLOBAL CONTEXT

The Legitimate Peripheral Participation approach [18] proposes that learning, i.e., the learners' participation in practice, is at first peripheral but increases gradually in engagement and complexity. In software projects, the practice is performing regular project tasks. It poses numerous challenges in globalized software development, for example, when developers attempt to integrate artifacts produced by heterogeneous tools, when they communicate with colleagues in a different time zone or speaking a different language. These challenges often lead to adverse results. For example, Mockus [22] found that one highly experienced developer may need up to six new replacement developers in the offshore location in large software projects, Herbsleb and Mockus [15] found that the time it takes to complete distributed tasks is almost three times longer than for co-located tasks.

Better understanding of how developers increase project competencies with practice is needed to resolve these issues. In the literature there are two proposed ways to address them. One is trying to understand empirically what hinders or helps the developers' learning. For example, Dagenias et al. [10] listed a number of obstacles facing developers joining new projects through studying 18 developers in IBM. Begel and Simon [2] found that communication and product knowledge pose serious challenges for the newcomers because they have not been trained for such tasks through their formal education. Some studies went further to find ways to facilitate globalization and learning. For example, Mockus and Weiss [24] proposed ways to separate large systems into independently changeable chunks to reduce dependencies among teams working on separate chunks. Sim and Holt [27] identified that mentors are an effective, though inefficient, way to teach newcomers in the project naturalization process.

The alternative approach is to investigate the differences between senior developers and novices. It has been argued that individual differences among project personnel accounts for the largest source of variation in project performance [9, 3]. Therefore explaining how the differences develop and change over time, with the hope of finding approaches that decrease the gap, appears to be a promising strategy. The representation of knowledge organization and development from cognitive science can provide an explanation for the nature of these differences. Therefore, cognitive science was used to understand the differences in mental structure between the experts and the novices in programming [9, 17]. Moreover, the striking differences were found that may be influenced by culture [19] and motivation [7].

Furthermore, various tools [23, 8, 1, 14] attempt to augment developers' knowledge of the project through visualizing or other ways of helping find the information developers might seek.

However, the understanding of how developers learn is in its infancy, therefore we will see a plethora of new ways to speed-up developers' learning and further understanding of how humans learn. As Fischer [13] suggested, learning and education are experiencing a period of profound transformation, the nature of problems is the systemic problems framed and solved by transdisciplinary collaboration.

3. UNDERSTANDING LEARNING

We consider three types of challenges posed by the need to understand and improve developer's project competencies.

First, the communication and coordination in conjunction with culture differences pose formidable obstacles to gaining project competencies in offshoring or outsourcing. Therefore, communication and culture are two critical factors that need to be studied in software project context.

Second, the study of the learning trajectory of the most productive developers, how they acquire their skills and resolve the central and complicated tasks, and what motivates them may provide externalization of the way they acquired expertise and be used for training others.

Third, the transfer from understanding to practice requires suggesting approaches, tools, and resources to help developers learn.

3.1 Culture and Communication

Work in social psychology, cognitive psychology, and anthropology suggests that all learning takes place in settings that have particular cultural and social norms and expectations and that these settings influence learning and knowledge transfer in powerful ways [4]. People in different cultures have strikingly different construals of the self, of others, and of the interdependence of the two. These construals

[2]We use the term *project competencies* to denote developers' ability to act in a particular situation and context. This depends on their knowledge, skill, and motivation, but the precise differentiation of how each of these concepts contribute to *project competencies* is beyond the scope of this paper.

[3]http://www.eclipse.org/mylyn/, http://code.google.com/p/hackystat/

can influence, and in many cases determine, the very nature of individual experience, including cognition, emotion, and motivation [19].

The fact that the practice should vary with culture has been perceived by the experienced people in the field. For example, an outsourcing manager in UK whose project is outsourced from UK to India said: "*There are different effects we must be aware of when dealing with an Indian team's attitude or culture and it is difficult to factor these in. For example, we expect a high level of attrition, so ownership has to be distributed among developers, which is not something we would do in the UK.*" "*From the India side, rapidly developing innovative products in an expedient manner is a less tangible skill.*"

Communication occupies a large part of a developer's programming life. As Begel et al. [1] found in Microsoft, majority of indicated developer needs involved discovering, meeting, and keeping track of people, not just code. However, there are many arguments that coordination poses a serious problem. As an outsourcing manager in China who gets outsourcing tasks from Japan complained: "*Often a very simple question needs to be bounced back and forth many times* (because of misunderstanding)." What are the developers' information needs? For example, Ko et al [16] identified the two types of most frequently sought information by developers that depended on their coworkers: "what have my coworkers been doing?" and "in what situations do these failures occur?" How developers communicate? An idea of Social Mechanism of Interaction introduced by Schmidt et al, emphasises the role of product itself in supporting the articulation of the distributed activities of multiple actors [26]. Not only codebase [11], but also bug report forms [5] are means by which the articulation work of the project can be carried out. Similarly, Zhou and Mockus [29] discovered the artifacts like MR/change repositories are important and sometimes the only possible mechanism for developers to communicate, in particular in the offshoring or transgeneration scenarios, since there may be no traditional opportunities to communicate in many offshoring situations, and a new generation of developers may be unable to communicate with original creators who have retired or died a long time ago. What is the intrinsic nature of communications? Nakakoji et al. [25] highlighted that the communication should not be regarded as something to be increased by itself, what matters is the quality of communication experience. In particular, the work introduced the concept of attention conservation inspired by the Internet- and gadget-paced lifestyle with people inundated with the flood of information and communication media. An example of improved quality of communications is described by Cataldo et al. [6] where the most productive developers, changed their use of electronic communication media over time, achieving higher congruence between coordination requirements and coordination activities.

Overall, the study of communication and its best practices in global and multicultural software development is a topic that would lead to better strategies to improve project competencies.

3.2 The learning of experts

What is known about experts is important not because all learners are expected to become experts, but because the knowledge of expertise provides valuable insights into what the results of effective learning look like [4].

First we need to understand the project practice trajectories that experts take. The issues include how a developer starts from a novice and becomes an expert, how she grows her expertise, and what kind of expertise she has to master (and in what order) to become central [28]. In particular, it's interesting to investigate how developers proceed from the role of a newcomer to the role of a core team member in the OSS projects, since there is an absence of a hierarchical organization, training plan, and a centralized environment for people to be nurtured. We gave some hints in [28] about how the developers grow their strength in terms of task difficulty and task centrality, but much broader and deeper investigation is needed, for example, of what leads to that trajectory.

Second, knowing how learners develop coherent structures of information has been particularly useful to understand the nature of organized knowledge that underlies effective comprehension and thinking. For example, as we suggested in [28], the difference between seniors and novices, might lie in the ability to combine and apply what is learned to perform more complex activities creatively and in new situations. Psychologists tried to aid software engineering through programmer selection testing since the 1950's. For example, McKeithan et al. [20] observed that experts are able to remember language commands based on their position in the structure of the language. Novices, not having an adequate mental representation of the language structure, often use mnemonic tricks to remember command names. Curtis [9] considered the performance of someone tackling a complicated programming task to be related to the richness of their knowledge about the problem area. However, the initial attempt had failed poorly, not because the principles and technologies of psychology were not up to the task, but because the psychologists failed to adequately model the mental and behavioral aspects of programming before selecting tests to measure it [9]. Learning theory can now account for how learners acquire skills to search a problem space and then use these general strategies in many problem-solving situations [4].

Third, the most influential factor to affect learning might be the motivation of a developer. In particular, the outsourcing, inhouse, and OSS developers are likely to be motivated and involved in the project for fundamentally different reasons. Motivation is likely to vary substantially even within outsourcing projects. For example, completely outsourced multi-year products are likely to elicit more commitment from developers in the outsourcing organization than projects outsourcing only a subset of the tasks or projects that last only a few months. For example, a developer for offshoring tasks from a site in India when being asked what could make him happier said: "*more new and interesting features*". This difference appears to play a crucial role in motivating people to learn, for example, an outsourcing manager in Romania said, "*we hire engineers who want to work in a particular area, and this provides motivation for them.*"

Overall, the advances in models of differences among individuals will come from a better understanding of the programmer knowledge base, and why and how the programmer learns.

3.3 Recommending system for learning

Recommender systems are currently attracting attention from software engineering researchers and practitioners. These are tools that help developers and managers better to cope with the huge amount of information faced in today's software projects, as noted in the announcement for the workshop on recommendation systems for software engineering[4].

There has been substantial amount of work to provide tools to improve developers' learning and performance in software projects. For example, Expertise Browser [23] presents the relationships between developers and source code they change to help determine experts for the code and developer expertise profiles, Hipikat [8] provides access to the group memory that is implicitly formed by all of the artifacts produced during the development for a software project, and Ariadne as a plug-in for Eclipse offers developers visualizations of the source code authorship and the potential presence of coordination problems [12]. More recently, the degree of knowledge model [14] adjusts developer expertise for code that was later changed by others, Codebook [1] tries to discover transitive relationships between people, code, bugs, test cases, specifications, and related artifacts by mining all kinds of software repositories.

We need better tools for the externalization and internalization of project knowledge, because:

- typically the externalization of knowledge is not the top priority in software projects, thus requiring non-intrusive individual and project activity recording tools with methods to extract meaning from such recorded traces of activities of unmotivated actors.
- the motivation to have a long-term learning of the project may be lacking, and necessitating tools and methods to make newcomers effective on the timescales of days or weeks not years.

According to Curtis [9], until the many sources of variation among individuals have been compared in the same set of data, it will not be possible to determine precisely which of the potential sources is the most important predictor of success in training programs or on the job. Also, as argued by Mockus [21], some fundamental questions can be answered only by considering the entire universe of publicly available source code and its history. All kinds of repositories and methods of analyzing them have sprung up recently to provide this kind of opportunity, for example, the data from a project's version history. More diverse and more detailed repositories are bound to appear with corresponding methods and tools to analyze them in the future. The research in this area will make it possible to measure aspects of human cognition in more detailed and more relevant ways and that will open new possibilities to understand and improve developer's project competencies to satisfy the increasing demands of the rapidly changing, multicultural, and global software development.

4. CONCLUSION

This paper proposes that the prevalence of offshoring, outsourcing, and open source development have posed critical challenges for developers to grow their project competence. Understanding the nature of culture differences and communication, investigating how experts learn and grow their expertise, and proposing resources and tools to facilitate the newcomers' learning, are promising opportunities to address challenges posed by these trends.

Improving this understanding has a particular significance in light of changes in what is to be expected of training newcomers in the global and multicultural software organizations that have to take on new projects at a moment's notice. Furthermore, the proposed understanding will not only borrow the developed concepts and approaches to be used in software engineering, but will also contribute to a more general scope — how humans learn.

[4]ICSE'10 Workshop, Cape Town, South Africa

5. ACKNOWLEDGMENTS

This work is supported by the National Basic Research Program of China under grant No. 2009CB320703, the Science Fund for Creative Research Groups of China under grant No. 60821003 and the Nature Science Fundation of China under grant No. 61073016.

6. REFERENCES

[1] A. Begel, K. Y. Phang, and T. Zimmermann. Codebook: Discovering and exploiting relationships in software repositories. In *ICSE '10: Proceedings of the 32th international conference on Software engineering*, pages 125–134, New York, NY, USA, 2010. ACM.

[2] A. Begel and B. Simon. Novice software developers, all over again. In *International Computing Education Research Workshop*, Sydney, Australia., 2008.

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

[4] J. Bransford, A. Brown, and R. Cocking. *How People Learn: Brain, Mind, Experience and School*. National Academy Press, Washington, D.C., 2003.

[5] P. Carstensen. The bug report form, 1994. http://cscw.dk/schmidt/papers/comic_d3.2.pdf.

[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] J. D. Couger and R. A. Zawacki. *Motivating and Managing Computer Personnel*. John Wiley & Sons, Inc., New York, NY, USA, 1980.

[8] D. Cubranic and G. Murphy. Hipikat: A project memory for software development. *TSE*, 31(6), 2005.

[9] B. Curtis. Fifteen years of psychology in software engineering: Individual differences & cognitive science. In *ICSE'84*, pages 97–106, 1984.

[10] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vrie. Moving into a new software project landscape. In *ICSE 2010*, pages 275–284, Cape Town, South Africa, May 1-8 2010.

[11] C. de Souza, J. Froehlich, and P. Dourish. Seeking the source: software source code as a social and technical artifact. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 197–206, New York, NY, USA, 2005. ACM.

[12] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles. Supporting collaborative software development through the visualization of

socio-technical dependencies. In *GROUP '07: Proceedings of the 2007 international ACM conference on Supporting group work*, pages 147–156, New York, NY, USA, 2007. ACM.

[13] G. Fischer. Cultures of participation and social computing: Rethinking and reinventing learning and education. *IEEE International Conference on Advanced Learning Technologies*, 0:1–5, 2009.

[14] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill. A degree-of-knowledge model to capture source code familiarity. In *ICSE '10: Proceedings of the 32th international conference on Software engineering*, pages 385–394, New York, NY, USA, 2010. ACM.

[15] 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.

[16] A. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *29thInternational Conference on Software Engineering*, pages 344–353. ACM Press, May 20–26 2007.

[17] J. Koenemann and S. P. Robertson. Expert problem solving strategies for program comprehension. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 125–130, New York, NY, USA, 1991. ACM.

[18] J. Lave and E. Wenger. *Situated Learning. Legitimate Peripheral Participation*. Cambridge University Press, Cambridge, 1991.

[19] H. R. Markus and S. Kitayama. Culture and the self: Implications for cognition, emotion, and motivation. *Psychological Review*, 98(2):224–253, 1991.

[20] K. McKeithen, J. Reitman, H. Rueter, and S. Hirtle. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13:307–325, 1981.

[21] A. Mockus. Amassing and indexing a large sample of version control systems: towards the census of public source code history. In *6th IEEE Working Conference on Mining Software Repositories*, May 16–17 2009.

[22] A. Mockus. Organizational volatility and its effects on software defects. In *ACM SIGSOFT / FSE*, Santa Fe, New Mexico, 2010.

[23] 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.

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

[25] K. Nakakoji, Y. Ye, and Y. Yamamoto. Comparison of coordination communication and expertise communication in software development: Motives, characteristics, and needs. In *Proceedings of JSAI-isAI2009 Workshop on KCSD2009*, pages 112–122. Springer Verlag, 2009.

[26] K. Schmidt and C. Simone. Coordination mechanisms: Towards a conceptual foundation of cscw systems design. *The Journal of Collaborative Computing*, 5:155–200, 1996.

[27] 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.

[28] M. Zhou and A. Mockus. Developer fluency: Achieving true mastery in software projects. In *ACM SIGSOFT / FSE*, Santa Fe, New Mexico, 2010.

[29] M. Zhou, A. Mockus, and D. Weiss. Learning in offshored and legacy software projects: How product structure shapes organization. In *ICSE Workshop on Socio-Technical Congruence*, Vancouver, Canada, May 19 2009.