1 Research Summary

I study software development projects and other complex systems through the recovery, documentation, and analysis of digital remains (i.e., *Digital Archeology*). These digital traces reflect various projections of collective and individual activity, but unlike traditional measurement systems, they represent a complex interplay among individuals, groups, their culture, and artifacts they produce. While it is straightforward to tabulate various measures obtained from digital traces, the direct interpretation of results is usually misleading. Only by integrating all relevant (and usually disparate) sources of data and disentangling contributions from multiple factors, can a meaningful inference take place. Thus, the fundamental method of *Digital Archeology* is the reconstruction and quantification of what actually happens from these projections.

Data Science — a recently popular term — is not clearly defined, but, according to common use, it primarily concerns business analytics, where the data and the interpretation of data are relatively straightforward, i.e., purchasing, click trails, and social media activity. My focus is on the methods that go well beyond that by combining disparate data sources including work, code, organizational, sales, and customer support histories over extended periods of time to construct rich, detailed models of events and people, and developing analysis methods that can identify the evolution and causes of events.

For example, user-reported defects (the most commonly used quality measure), are predicated on the existence of a user engaged in a task utilizing software, on that user observing a failure and then being willing and able to report it in detail, and on the product maintainer being interested and able to fix the defect causing that failure. Because this long chain of events has negative feedback loops (users stop using software if the chances of observing a defect become too high), the defectiveness of software perceived by users is often negatively correlated with defect density. A meaningful measure of software quality must, thus, account for the number of users and the amount of usage.

While using digital traces as a measurement system is enormously complex, such analysis provides a practical, unmatched ability to conduct live, non-intrusive, and fine-grained way to gain deep insight into organizational processes. Using this approach provides a unique capacity to observe, analyze, and support software development and other complex socio-technical systems [1]. The analysis may focus on entire companies [2, 3] or an entire population of open source projects [4, 5].

2 Impact

I significantly contributed to the field of software measurement. In particular, my focus on using code changes [6, 7] as fundamental units of software development — the critical interface between the code and the person modifying it — provided a methodology to obtain valid and relevant (in practice) measures and models based on operation support systems, such as Version Control Systems (VCS) and Issue Tracking Systems (ITS). In my early work I demonstrated how to use software changes obtained from VCS and ITS as a basic measurement unit and that proved to be a solid foundation to build upon. In, particular, I showed how to obtain, classify and measure [7] software changes obtained from VCS, how to use them for fine-grained estimation of effort [8, 9], measure expertise [10] and interdependencies [11, 12], predict defects [13, 14], and, more generally, to provide insight into software development practices [1, 15]. The quantification of open source development practices [1] had impact in domains not related to software engineering, as shown for example, by extensive citation of my work in diverse fields such as innovation [16], collaborative work [17], and management science [18], among others.

From the outset, my measurement and modeling was focused on having a practical impact, and that led to the choice of operation support systems to serve as a measurement tool. This allowed me to help improve software development in Lucent and Avaya by developing valid measures of software quality, by establishing clear evidence of the value brought by basic software development practices, and by deploying innovative procedures and tools to improve software development. In particular, by combining insights from software development models and software change data, I have constructed tools optimally to distribute work [19], estimate and present developer expertise [10], and improve quality of software [20, 21]. Customer Quality Measure (CQM) — the fraction of customers who encounter defects within three months of installation — is used to set quality goals for all major Avaya projects. The empirical relationships between basic practices (inspection/testing/static analysis) and CQM, has led to a wide deployment of these practices within Avaya and, in turn, led to significant improvements in CQM saving more than US$1.5M [1] in a single project. A case study of another project looking at the effects of applying risk mitigation activities to the one percent of project's code where most of customer reported issues were fixed [22], showed US$300K savings in prevented development effort. The risk mitigation approach has been, so far, deployed to over 30 Avaya projects.

[1] Of prevented development effort related to customer issues.

3 Vision

Software development is not an isolated activity: apart from dealing with the technology, individuals, and teams, it also has to satisfy existing needs, generate new needs, and react to technical, economic, and political changes. These larger forces shape and direct software development in new ways as in, for example, outsourcing/offshoring and cloud computing. The issues facing software development have to be resolved within this broader context. Furthermore, other forms of human endeavor, such as commerce, support activities, and games, are conducted in the digital realm leaving traces of unprecedented detail and generating an explosion of interest in ways to utilize such traces [23].

Cloud computing opens new possibilities for *Digital Archeology* by concentrating immense collections of software development traces (Googlecode.com has more than 300 thousand software projects and Github.com has more than six million repositories). Mobile computing and social networks enrich the traces with the explicit geographic and social components.

At the largest scale I will derive insights into the behavior of the entire open source and corporate software development and of related socio-technical systems. I will create a repository of open source, corporate, and government software project data, create valid measures that describe individual and organizational behavior at this level. I have already obtained the collection of most publicly available source code version history data [4], and I have started a collaboration with Peking University in China and Queens University in Canada to collect other sources of data such as problem tracking systems and mailing lists. From the corporate perspective, I will create a multi-company repository similar to the one I created for Avaya [3] to study software and organizational phenomena in a global setting [24, 25]. Such global questions start from basic census: what is the distribution of code, people, and their activities and how does it depend on context? Higher level questions would include comparison of cultures (for example by comparing the creation of Wikipedias in different languages) and the spread of innovative practices and artifacts. The answers to such questions would provide a basis for engineering socio-technical systems with desired features, for example, longevity.

At the intermediate scale I will study related groups of projects (ecosystems) in the open source and commercial environments. A key open question is how do ecosystems come into being? One hypothesis is that a successful project will grow until a core team is so large that it can no longer cope with the increased amount of coordination. Another critical question is to what extent the context determines the best practices of software development [26, 27]. For example, as a result of acquisitions, Avaya has multiple products that serve identical business purposes and each uses its own development and support practices: to what extent are these practices determined by the company, the technical structure of the product, or the customer requirements? While the project context may be a crucial factor that determines the most suitable tools and practices, experiments that could control for the context in real software projects are rarely conducted [26, 28] due to extremely high costs. When conducted, however, they bring surprises, e.g., only the overall systems size affects maintenance effort [29]. Natural experiments [2] where different companies or open-source projects produce software with similar functionality would be possible using multi-company data described above.

At the smallest scale I will study individuals' micro-activity while navigating or editing artifacts and interacting with others. Such data can be obtained via plug-ins of the authoring environments, e.g., Eclipse, from frameworks that support analysis of web activities, e.g., Google Analytics, from sensors of mobile and, in the future, of wearable devices. Such micro-level data could dramatically increase the power of the higher level models to predict individual's actions and motivation.

To solve the outlined challenges, it will be essential to refine and enhance *Digital Archeology* in several ways. Technical challenges include scaling to larger and more diverse data. Analytic challenges include piecing together these unstructured, dirty, and difficult-to-combine data. Novel analysis and visualization techniques will be needed to address these demanding tasks.

In summary, software development involves people, teams, organizations, culture, and society. My research, therefore, will both learn from and, also, will contribute to these domains to the extent that software development mirrors any other human endeavor. By clarifying ways in which software development is conducted it may, therefore, help make progress answering the basic questions about how people organize themselves and their work.

[2] A natural experiment is an empirical study in which the experimental conditions are out of the control of the experimenters but treatment assignment is, arguably, random. For example, studies of identical twins separated at birth to determine relative impacts of genetic vs environmental impact on development.

4 Research Topics

Cost and quality

More than a decade ago, I introduced a methodology that uses widely available repositories of automatically recorded project data in change management and problem tracking systems to model and analyze software development. By performing statistical analysis of software changes I quantified the most influential drivers of cost [30, 31], interval [11], quality [20] and differences between development patterns in commercial and open source software development [15]. These methods are now widely used throughout the software engineering community, for example, most defect prediction work uses historic changes and defects as in, for example, [32].

The impact of my work dramatically affected practice because the information crucial for decision making could finally be obtained without incurring prohibitive costs and delay of manual collection practices [33], thus leading to better and radically different decisions and practices as outlined below. The impact was not contained within a single company, but affected the entire industry, including companies such as Microsoft, IBM, AT&T, and Alcatel/Lucent.

In particular, I developed techniques for precise estimation of productivity gains for a variety of tools and software development methods. This led to their adoption and to changes in development practices as described in, e.g., [34, 35, 3].

The investigation of quality issues in software patches resulted in the discovery that the desire to increase the revenue by including new features in patches made it inevitable that such patches would fail (as the new features greatly increase the size of patches, and statistical analysis of failures predicts that sufficiently large patches would fail.) This led, among other things, to the abandonment of this apparently lucrative, but harmful practice [36, 20].

Large-scale phenomena

Assembling large, interconnected data sets from multiple projects and data sources allows us to answer entire classes of questions about large-scale behavior that could not be addressed in other ways. I created methods for discovering, acquiring, integrating, and analyzing these heterogeneous types of data [37, 38]. A complete collection of data within a company [3] or an entire collection of open source code [4], gives us ability to determine the source code origins and authorship via Universal Version History [39, 40] or the spread of innovation via code reuse [5]. The authorship and code origin analyses are used in Avaya to identify open source code, thus addressing key source code licensing issues. More importantly, by helping to understand who creates code and how (and by whom) it is later (re-)used in other projects, such analysis opens the possibility to investigate creativity and innovation in software development: an intangible asset that is both crucial to business success and, at the same time, is prone to be the first casualty in any cost-cutting, offshoring, or outsourcing scenario.

Transfer of work

Software development practice is experiencing a radical change driven by the open source movement, the business needs to move development to low-cost locations, the aging and renewal of core developers in legacy products, and recruiting in fast growing Internet companies, all of which are causing unprecedented turnover in software projects.

To address these challenges, my research investigates the transfer of work [41] and the associated phenomena related to organizational change [2] and the growth of software project competencies [42]. More specifically, I discovered that the relationship between the social and technical competencies was associated with the fraction of new participants who become long term contributors of a software project [43]. It provides a tantalizing opportunity to study how the initial environment a person encounters when joining a team or an organization affects motivation and long-term behavior.

The findings of performance issues related to transfer of code [41] and the project expertise [42] led to radical changes in the way work transfers were handled, in improvement of the hiring practices at offshore locations, and other substantial changes.

Given the diversity of software projects and the creative nature of a software development, I sought to find out how much the context of a project may affect the outcomes by conducting multi-company studies [26, 25, 24] which took into account not only technical, but also social [12, 24] aspects of software development. Surprisingly, a software project with identical requirements may cost an order of magnitude more and require correspondingly more effort simply because of the differences in the nature of company's customer base [26]. This variation can not be explained by extant effort estimation methods and needs more research, perhaps using twin-project approach outlined above, to understand and quantify the reasons for the differences between products with identical functionality and to develop suitable estimation tools. At the same time, many phenomena related to how developers make mistakes leading to software defects are similar in diverse projects and companies [2, 24, 25].

User interfaces, visualization, optimization

I began developing the methods that became *Digital Archeology* in my earlier work in which I investigated user interfaces [44, 45] and the visualization and modeling of complex phenomena such as the spread of diseases [46, 47], displays of a large number of images [48], and the response of the brain to visual or cognitive stimuli [49, 50]. My work on algorithms involves Bayesian methods of optimization for non-convex functions [51], boosting methods to optimize parameters of discrete optimization heuristics [52], clustering of high-dimensional datasets [53], isotonic regression [54], and the estimation of a covariance function from irregularly-shaped spatial aggregates [47].

Way forward

To achieve my goals I will use my diverse skills and, where needed, I will forge collaborations to address critical practical issues at different scales and in various contexts, attempting to paint a unified picture of individual and collective behavior and enabling progress at the critical junction of society and technology. The topics outlined above are but examples of what could be achieved by using increasingly prevalent digital traces for all kinds of phenomena. I will, therefore, focus on refining fundamental methods of high-integrity inference from low-quality sources generated by a variety of operational systems.

References

[1] 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.

[2] Audris Mockus. Organizational volatility and its effects on software defects. In *ACM SIGSOFT / FSE*, pages 117–126, Santa Fe, New Mexico, November 7–11 2010.

[3] Randy Hackbarth, Audris Mockus, John Palframan, and David Weiss. Assessing the state of software in a large enterprise. *Journal of Empirical Software Engineering*, 10(3):219–249, 2010.

[4] Audris 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.

[5] 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.

[6] Audris Mockus, Todd L. Graves, and Alan F. Karr. Modelling software changes. In C.E. Minder and H. Friedl, editors, *Good Statistical Practice*, pages 175–179. Austrian Statistical Society, Wien, Austria, July 1997. Proceedings of the 12th International Workshop on Statistical Modeling, Biel/Bienne.

[7] Audris Mockus and Lawrence G. Votta. Identifying reasons for software change using historic databases. In *International Conference on Software Maintenance*, pages 120–130, San Jose, California, October 11-14 2000.

[8] Todd L. Graves and Audris Mockus. Inferring change effort from configuration management data. In *Metrics 98: Fifth International Symposium on Software Metrics*, pages 267–273, Bethesda, Maryland, November 1998.

[9] D. Atkins, T. Ball, T. Graves, and A. Mockus. Using version control data to evaluate the effectiveness of software tools. In *1999 International Conference on Software Engineering*, pages 324–333. ACM Press, May 16–22 1999.

[10] 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.

[11] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. An empirical study of global software development: Distance and speed. In *23nd International Conference on Software Engineering*, pages 81–90, Toronto, Canada, May 12-19 2001.

[12] James Herbsleb and Audris Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *2003 International Conference on Foundations of Software Engineering*, Helsinki, Finland, October 2003. ACM Press.

[13] Audris Mockus, David M. Weiss, and Ping Zhang. Understanding and predicting effort in software projects. In *2003 International Conference on Software Engineering*, pages 274–284, Portland, Oregon, May 3-10 2003. ACM Press.

[14] Audris Mockus. Analogy based prediction of work item flow in software projects: a case study. In *2003 International Symposium on Empirical Software Engineering*, pages 110–119, Rome, Italy, October 2003. ACM Press.

[15] Audris Mockus, Roy T. Fielding, and James Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1–38, July 2002.

[16] Georg von Krogh, Sebastian Spaeth, and Karim R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, July 2003.

[17] Gerard Beenen, Kimberly Ling, Xiaoqing Wang, Klarissa Chang, Dan Frankowski, Paul Resnick, and Robert E. Kraut. Using social psychology to motivate contributions to online communities. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 2004.

[18] Alan MacCormack, John Rusnak, and Carliss Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Journal Management Science*, 52(7), 2006.

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

[20] Audris Mockus and David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[21] Audris Mockus and David Weiss. Interval quality: Relating customer-perceived quality to process quality. In *2008 International Conference on Software Engineering*, pages 733–740, Leipzig, Germany, May 10–18 2008. ACM Press.

[22] Audris Mockus, Randy Hackbarth, and John Palframan. Risky files: An approach to focus quality improvement effort. In *9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2013.

[23] Nathan Eagle, Michael Macy, and Rob Claxton. Network diversity and economic development. *Science*, 328(5981):1029–1031, May 2010.

[24] Marcelo Cataldo, Audris Mockus, Jeffrey A. Roberts, and James D. Herbsleb. Software dependencies, the structure of work dependencies and their impact on failures. *IEEE Transactions on Software Engineering*, 2009.

[25] Audris Mockus, Nachiappan Nagappan, and T Dinh-Trong, Trung. Test coverage and post-verification defects: A multiple case study. In *International Conference on Empirical Software Engineering and Measurement*, Lake Buena Vista, Florida USA, October 2009. ACM.

[26] Bente C.D. Anda, Dag I.K. Sjøberg, and Audris Mockus. Variability and reproducibility in software engineering: A study of four companies that developed the same system. *IEEE Transactions on Software Engineering*, 35(3), May/June 2009.

[27] Minghui Zhou, Audris Mockus, and David 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.

[28] Audris Mockus, Adam Porter, Harvey Siy, and Lawrence G. Votta. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering and Methodology*, 7(1), January 1998.

[29] Dag I.K. Sjøberg, Bente Anda, and Audris Mockus. Questioning software maintenance metrics: a comparative case study. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '12, pages 107–110, New York, NY, USA, 2012. ACM.

[30] D. Atkins, A. Mockus, and H. Siy. Measuring technology effects on software change cost. *Bell Labs Technical Journal*, 5(2):7–18, April–June 2000.

[31] D. Atkins, T. Ball, T. Graves, and A. Mockus. Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering*, 28(7):625–637, July 2002.

[32] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(2), 2000.

[33] R Grady and E Caswell. *Software metrics*. Prentice-Hall, Englewood Cliff, 1987.

[34] Birgit Geppert, Audris Mockus, and Frank Rößler. Refactoring for changeability: A way to go? In *Metrics 2005: 11th International Symposium on Software Metrics*, Como, September 2005. IEEE CS Press.

[35] D. L. Atkins, A. Mockus, and H. P. Siy. *Value Based Software Engineering*, chapter Quantifying the Value of New Technologies for Software Development, pages 327–344. Springer Verlag Berlin Heidelberg, 2006.

[36] A Mockus, H Siy, T Sundresh, J Chen, and TL Graves. Role of change size, complexity, and developer expertise in predicting the quality of a software update. Technical Report 10009677-000324-01TM, Lucent Technologies, 2000.

[37] Audris 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.

[38] Audris Mockus. Missing data in software engineering. In J. Singer et al., editor, *Guide to Advanced Empirical Software Engineering*, pages 185–200. Springer-Verlag, 2008.

[39] Hung-Fu Chang and Audris Mockus. Evaluation of source code copy detection methods on FreeBSD. In *5th Working Conference on Mining Software Repositories*. ACM Press, May 10–11 2008.

[40] Hung-Fu Chang and Audris Mockus. Constructing universal version history. In *ICSE'06 Workshop on Mining Software Repositories*, pages 76–79, Shanghai, China, May 22-23 2006.

[41] Audris Mockus. Succession: Measuring transfer of code and developer productivity. In *2009 International Conference on Software Engineering*, Vancouver, CA, May 12–22 2009. ACM Press.

[42] Minghui Zhou and Audris Mockus. Developer fluency: Achieving true mastery in software projects. In *ACM SIGSOFT / FSE*, pages 137–146, Santa Fe, New Mexico, November 7–11 2010.

[43] Minghui Zhou and Audris Mockus. Does the initial environment impact the future of developers? In *ICSE 2011*, pages 271–280, Honolulu, Havaii, May 21–28 2011.

[44] Stacie Hibino and Audris Mockus. handiMessenger: Awareness-enhanced universal communication for mobile users. In Fabio Paternò, editor, *Mobile Human-Computer Interaction, 4th International Symposium, Mobile HCI 2002, Pisa, Italy, September 18-20, 2002, Proceedings*, volume 2411 of *Lecture Notes in Computer Science*, pages 170–183, Pisa, Italy, September, 18-20 2002. Springer.

[45] S.L. Hibino, T. Graves, and A. Mockus. A web based approach to interactive visualization in context. In *Advanced Visual Interfaces*, pages 181–188, Palermo, Italy, May 23-26 2000.

[46] W.F. Eddy and A. Mockus. An example of the estimation and display of a smoothly varying function of time and space - the incidence of mumps disease. *Journal of the American Society for Information Science*, 45(9):686–693, 1994.

[47] Audris Mockus. Estimating dependencies from spatial averages. *Journal of Computational and Graphical Statistics*, 7(4):501–513, 12 1998.

[48] W.F. Eddy and A. Mockus. An interactive icon index: Images of the outer planets. *Journal of Computational and Graphical Statistics*, 5(1):100–111, 1996.

[49] A. Mockus, W.F. Eddy, S.Y. Chang, and K.R. Thulborn. Software for the visualization of fMRI data. In *Proceedings of the International Society for Magnetic Resonance in Medicine Fourth Scientific Meeting and Exhibitionn*, page 1774, 1996.

[50] W.F Eddy, M. Fitzgerald, C. Genovese, N. Lazar, A. Mockus, and Welling J. The challenge of functional magnetic resonance imaging. *Journal of Computational and Graphical Statistics*, 8(3):545–558, September 1999.

[51] A. Mockus and L. Mockus. Designing software for global optimization. *Informatica*, 1(1):71–88, 1990.

[52] A. Mockus, J. Mockus, and L.Mockus. Bayesian heuristic approach (BHA) and applications to discrete optimization. *Fields Institute Communications*, 18:153–165, 1998.

[53] W.F. Eddy, A. Mockus, and S. Oue. Approximate single linkage cluster analysis of large data sets in spaces of high dimension. *Computational Statistics and Data Analysis*, 23:29–43, 1996.

[54] M. Lavine and A. Mockus. A nonparametric Bayes method for isotonic regression. *Journal of Statistical Planning and Inference*, 46:235–248, 1995.