

# Statement on Research, Teaching and Service

Premkumar Devanbu

<http://www.cs.ucdavis.edu/~devanbu>

## 1 Teaching

I find every aspect of teaching programming and software engineering immensely challenging and rewarding: the narrative craft of lecturing, one-on-one interactions with students (both struggling and talented), preparing assignments, and providing evaluative feedback to students.

**Coursework:** I teach ECS160, a senior capstone project class in software engineering. Students learn software engineering practice, grounded in a large, complex team project which follows a prescribed prototyping lifecycle. I individually meet with teams of 4-5 students several times during the quarter, to aid in project definition, user-interface design, and team social dynamics. Classroom pedagogy is focused strongly on design, and process, using patterns at every level, from object-oriented design patterns and architectural styles to process models. Recent evaluations have been over 9 (on a 10-point scale), despite rigorous grading (only about 10% of students receive A- or better). Gratifyingly, many students state that my version of 160 requires more work than any other course in the curriculum, but also profess their enjoyment of the material.

In my graduate software engineering class, ECS260, I present advanced techniques of programming, (meta-programming, partial evaluation, reflection, aspects, code generation, functional programming) illustrated in several mini-projects. I have also been running another graduate seminar on empirical software engineering and open-source development.

**Graduate Supervision:** The challenge for me in graduate supervision is the creation of a good *lab culture*: ideally, a vibrant, fertile intellectual environment where students actively and constantly engage each other and the most important current concepts in the field. In such a culture, students critique, support and learn from each other. The lab becomes a place where people want to hang out, and a focal point for idea generation and problem solving. The advisor's role is merely to be coach, gadfly and cheerleader. It's been gratifying to nurture this type of culture in our lab, and observe it begin to bear fruit.

## 2 Research

My primary research interest is in empirical software engineering, using both open-source and commercial software. I also have interests in software tools.

### 2.1 Empirical Software Engineering

The bioinformatics field exploded 15 to 20 years ago when microarrays and gene-sequencing devices started producing torrents of data. Now, with the vast treasure trove of archived data from open source software (OSS), empirical software engineering is in a similar state of foment. Various important claims, whose status thus far haven't progressed much beyond folklore, can now be more rigorously studied.

**Social aspects of software engineering** Real software engineering is an inherently collaborative activity. Successful projects require well-organized teams, effective communication, productive

collaborations, and smooth incorporation of newcomers. Remote collaboration is increasingly important (and problematic). Fortunately, data pertinent to these matters is now available from open-source archives, as well as in some commercial projects. This has led to several results.

We have studied communication and collaboration patterns, and the effect of remote collaboration. We have shown tight relationships between communication effort and development activity in open-source projects [2, 11, 12]. We have also studied the absorption of newcomers into successful open-source projects, and found that conflicting trends lead to a non-monotonic rate of immigration [8]. We have found that open-source projects, often believed to be somewhat disorganized and bazaar-like, do in fact show significant levels of spontaneous self-organization [1] strongly relating to collaborative activities.

We have also studied the effect of remote collaboration on software processes. It has been generally believed that remote collaboration leads to less effective processes. Using data from an industrial partner, we find that this isn't always true: the quality of artifacts produced by teams is virtually unaffected by the geographical distribution of the team; we found this to be true even when considering the size and complexity of the artifacts [3]. This appears to be due to the rigorous collaborative processes followed in this company, and further study is underway

**Reliability of Quality Data** Proper deployment of scarce quality-control resources (inspection, rigorous testing, fixing static analysis warnings) strongly depends on reliable prediction of fault-prone elements of systems. Prediction techniques leverage past reports of bug fixes to fit prediction models, using statistical or machine-learning approaches. Unfortunately, not all bug fixes are reported; thus, existing approaches all implicitly assume that reported bug fixes are an unbiased, fair sample of the population of bug fixes. We have been studying whether this assumption holds. So far, we have found that there are actually several types of bias: fixes of more severe bugs are *less* likely to be reported; more experienced developers are *more* likely to report bug fixes; reviewed and verified bug fixes are *more* likely to be reported. Work is ongoing, to verify if other types of bias may exist. Preliminary results were presented in a poster at FSE 2008 in Atlanta.

## 2.2 Software Tools

Our current work is in the area of platforms for distributed, co-operative regression testing; most recent earlier work has been in recommender systems, and in middleware.

**Recommender Systems** *Recommender tools* help developers deal with information overload: given an item of interest, related items are retrieved. This can help find relevant code, and help gain familiarity with novel and complex APIs. With Prof. Filkov and Zach Saul (student) we adapted Kleinbergs' HITS algorithm to create a recommender tool which finds closely related functions given a function of interest, using a single version of the callgraph. We introduced a rigorous statistical evaluation regime, and found that it outperformed existing tools. Our paper on this tool [7] was nominated for an ACM Distinguished paper award.

**Middleware** *Middleware* is a software layer above the OS which provides convenient abstractions for application developers. These abstractions mask details (*e.g.* distribution), thus simplifying programming. This abstraction layer, however, leads to poor performance in cases where the full power of the middleware is not needed. Our MOBYL (MOdel-driven BYpassing of middleware Layers) work addresses this problem: programmers can bypass middleware selectively. MOBYL allows applications to model bypassing applications at the IDL level, and is implemented using code generation tools and run-time enhancements. A MOBYL developer can “have her cake and eat it

too”: *viz.* achieve improved performance by passing unwanted layers of the middleware, while still enjoying type-safe programming and IDL abstractions [6].

**Distributed, Co-operative Regression Testing** Popular, feature-rich apps such as Office suites are constantly evolving, in response to market pressures and customer demand. Continuing correct function of old features is a vital yet challenging problem in this regard, requiring extensive regression testing prior to a new release. Failures in old features are costly and frustrating to customers, and can result in loss of reputation and market share. However, vendors often aren’t aware of the rich (or perverse) ways customers utilize products. We are developing a new approach, (TIGRESS, for Transparent, Internet-scale reGRESSion testing) wherein new releases are silently and co-operatively regression tested by customers, alongside old releases, during normal use, to check if the behaviours match. There are several challenges: performance, privacy, and large-scale data mining. In our experiments, we can reduce the customer performance overhead to a few percent; we are working (in co-operation with Microsoft Research) towards scaling up this approach and making it practical.

### 3 Impact & Recognition

I enjoy working in different areas, and interacting with colleagues from different intellectual disciplines. (both within and outside of computer science). Nevertheless, I have always tried hard to produce pertinent, interesting, original results in the different areas. Over the years, I have written well-cited papers that have attracted well over 100 citations in a number of different areas: knowledge-based software engineering, software tools, middleware, empirical software engineering, and security. I have also been invited to give keynote presentations, tutorials and distinguished lectures in several areas: empirical software engineering, knowledge representation, secure software engineering, and middleware. Within my main field (software engineering), the work has been well-received: my ICSE 1991, 1992, and 1997 papers were all nominated for 10-year most influential paper award; though, none actually did win the award. In addition papers at ICSE 2003 and SIGSOFT 2007 were nominated for a ACM SIGSOFT distinguished paper award, which we actually won at ICSE 2004.

In a recent (June 2007) ranking of academic software engineers published in *CACM* [4] I was ranked among the top software engineering researchers in the US, at position 8 (18<sup>th</sup> worldwide).

### 4 Service

**Journals:** Currently, I’m on a second term on the editorial board *The IEEE Transactions on Software Engineering*. I also served previously on the editorial board of *ACM Transactions on Software Engineering and Methodology*.

**Conferences:** I was PC Chair of ACM SIGSOFT FSE 2006, and will serve as co-PC chair for ICSE 2010. I have served on numerous occasions on PCs of these and other conferences.

### References

- [1] Christian Bird, David Pattison, Raissa DSouza, Vladimir Filkov and Premkumar Devanbu, Latent Social Structure in Open Source Projects, *Proceedings, Foundations of Software Engineering Conference*, 2008.
- [2] Pattison, D., Bird, C., Devanbu, P, Talk and Work: A preliminary report *Proceedings, 5th Annual Workshop on Mining Software Repositories*, 2008

- [3] Bird, C., Nagappan, N., Devanbu, P., Gall, H., and Murphy, M., Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista *Proceedings, ICSE 2009*
- [4] Ren, J., and Taylor, R.W., Automatic and versatile publications ranking for research institutions and scholars, *Communications of the ACM*, 2007
- [5] Static Checking of Dynamically Generated Queries in Database Applications. Gary Wassermann, Carl Gould, Zhendong Su, and Premkumar Devanbu. *Invited submission to ACM TOSEM*, accepted November 2006.
- [6] Demir, O., Devanbu, P., Wohlstadter, E., and Tai, S. An Aspect-oriented Approach to Bypassing Middleware Layers. In *Proc. of the International Conference on Aspect-Oriented Software Development*, 2007.
- [7] Z. Saul, V. Filkov, P. Devanbu, and C. Bird, Recommending RandomWalks, *ACM ESEC/SIGSOFT Foundations of Software Engineering*, 2007
- [8] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu, Open Borders? Immigration in Open Source Projects, *4th Annual Workshop on Mining Software Repositories*, 2007
- [9] C. Bird, A. Gourley, and P. Devanbu, Detecting Patch Submission and Acceptance in OSS Projects, *4th Annual Workshop on Mining Software Repositories*, 2007
- [10] M. Ogawa, K-L. Ma, C. Bird, P. Devanbu, A. Gourley, Visualizing Social Interaction in Open Source Software Projects, *Asia Pacific Symposium on Visualisation*, 2007
- [11] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, Mining Email Social Networks, *3rd Annual Workshop on Mining Software Repositories*, 2006
- [12] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, Mining Email Social Networks in Postgres” *3rd Annual Workshop on Mining Software Repositories*, 2006
- [13] Briand L., Devanbu, P., Melo, W., An Investigation into Coupling Measures for C++, *19th International Conference on Software Engineering*, 1997.