

## The Miracle of Open Source

Twenty years ago, if you told people that software developed by world-wide teams of volunteers would be giving Steve Ballmer and Bill Gates sleepless nights, you would have been placed in protective custody. Today, we routinely use (and marvel at the success of) Mozilla, Apache, Linux, MySQL etc. But there's another, quieter miracle going on: Open Source software (OSS) has initiated nothing less than a revolution in Software Engineering. For years, software engineers have developed theories, tools, and processes and lamented over the difficulty of testing the theories, validating the tools, or trying out the processes, since no one had access to data on how software engineers actually worked. Academics didn't have any data, and commercial developers were too busy and/or too paranoid/myopic (with very few exceptions) to share the data. Enter OSS project, where all activities are conducted in public on the internet, and which inherently expose complex, comprehensive, longitudinal narratives on what people do, how they communicate, how they fix bugs, when they release software, how many people download them etc. There's lots opportunities to: build tools to get the data, find ways to analyze the data, find new theories to test. It's an amazing time in software engineering, rather like in Biology, when the first high-throughput instruments like gene sequencers and Affymetrix micro-arrays came out. Finally we can ask (and provide real answers to) questions like: how do software teams really work together? how/why are defects introduced? How do software developers and projects stay innovative? How does software actually evolve? Which bugs get fixed quickly, and which take longer? What makes highly productive developers so productive? etc.

This seminar is intended to provide a broad introduction to open-source source, empirical software engineering,

### Instructor:

Prem Devanbu

[devanbu@cs.ucdavis.edu](mailto:devanbu@cs.ucdavis.edu)

### Pre-requisites

An undergraduate software engineering course, similar to ECS 160. Basic comfort level with SQL, probability, statistics, and linear algebra. Interest in Empirical methods.

### Expectations:

- I. Read 2 to 4 papers per week.
- II. Attend all classes and participate vigorously in discussions. (20% of credit)
- III. Prepare a presentation and lead classroom discussion on two topics (by arrangement with instructor). Prepare a moderate-length, insightful survey paper on one or both of the two topics. (80 % of credit)

## Possible Topics (not complete, will evolve)

- 1) The Dogma Open Source. Eric Raymonds essays, Weber, Economy of Open Source (by Instructor)
- 2) It's the Data, stupid. Flossmole etc. (by Instructor)
- 3) Software Metrics. (by Instructor]
- 4) Design, modularity. (by Instructor)
- 5) The Very Basics of data analysis in R, (guest lecture, or tutorial)
- 6) Statistical Physics of code--distributions. and theories. Newman, Sole, Garth Baxter. Log Normal distributions.
- 7) Finding and Studying Code Clones.
- 8) Mining sources and MR's for defects: PR-Miner, Hollingsworth, MAPO, Zeller, Zimmerman, Jiang, etc.
- 9) Recommender systems. Suade, etc., Brin, Hipikat, Zimmerman (cvs based guiding)
- 10) Software Evolution theory and data extraction and mining.
- 11) Social structures, networks. basics of social networks, and different types of networks. socio-technical congruence. Organizational theories of socio-technical congruence (e.g., Conway's law).
- 12) Immigration. of New Developers.
- 13) Visualization of Open Source Data.
- 14) Micro-design and recovery: graphlets, patterns, relations to bio-informatics.
- 15) API recovery: MAPO, etc.
- 16) Macro Design recovery: call graph clustering, and validation of recovered design.
- 17) Recommender Systems: Hipikat, Suade, FRAN.
- 18) Open-source Innovation, communities of practice, sociology of innovation.
- 19) Business Issues: How open source and commercial software interact.
- 20) Cognitive studies of programming activity.