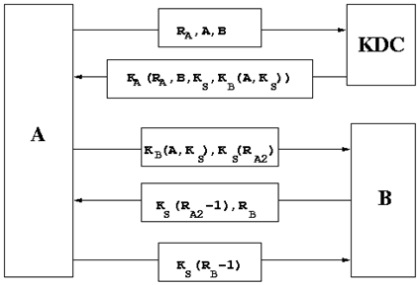
dB-SERC Mentor-Mentee Evidence-Based Teaching Award

Adam J. Lee
William C. Garrison III

Designing and building secure systems is hard!

"The black art of programming Satan's computer" [1]

Longstanding designs and implementations have been proven insecure:



Needham-Schroeder
Man-in-the-middle discovered after 17 years in use

OpenSSL
Heartbleed vulnerability discovered 2 years after introduced

Formal verification is very difficult, even for experienced software engineers!

CS 1653 teaches security engineering with a focus on a semester-long group project

CS 1653: Applied Cryptography and Network Security

Lectures present algorithms and protocols, students apply these in an interleaved semester project

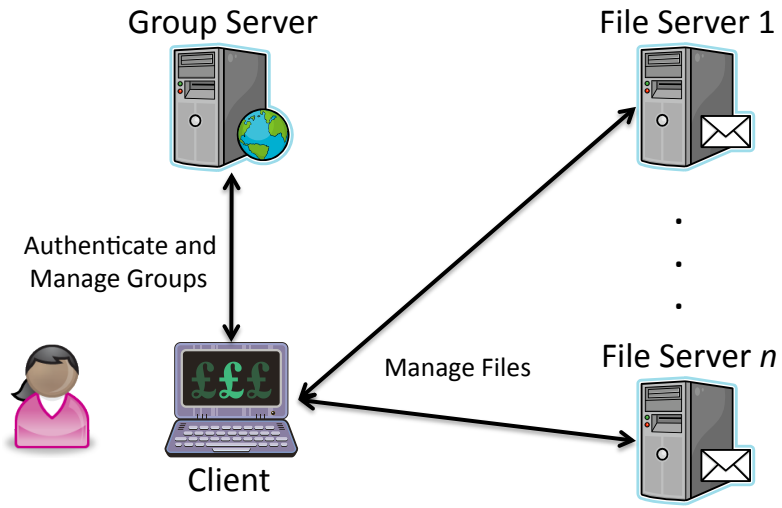
In this project, students must:

- Work in groups for the full semester
- Propose their own solutions to adversarial tasks
- Develop, maintain, and extend a non-trivial code base (~5k lines)

Requires both design and coding!

A summary of the CS 1653 semester project

Students develop a secure distributed file-sharing system



Five phases, each considering additional security threats

Students meet with instructor to propose solutions, demo with TA after submission

Even the best students run into problems with this project…

The most common problems:

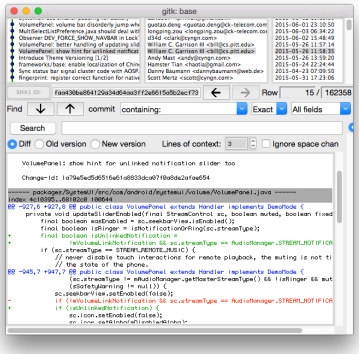
- Uneven distribution of work
- Lack of communication among group members
- Procrastination, submitting last-minute
- Juggling design and code
- Rushing through code
- Combining code written by multiple group members
- Design and code not matching, evolving out-of-sync

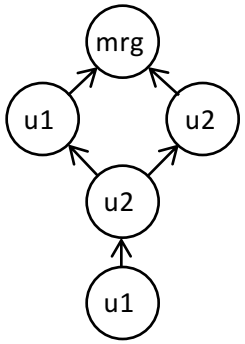
Can using a version control system help mitigate these issues?

Why develop code using a version control system?

In a VCS, any change to a code base is called a commit

The VCS maintains a history of previous commits with descriptions





A commit is relative, to ease the merging of work from multiple users

Commit logs are *time series* describing development at a fine granularity, and have been used for a variety of experiments:

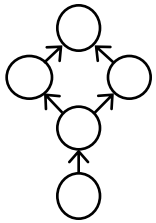
- Adoption of new APIs does not keep pace with their development [2]
- Programming language design has a modest effect on code quality [3]
- Gender and tenure diversity are positive predictors of productivity [4]
- Functions with asserts have significantly fewer defects [5]
- Asking questions on Q&A sites catalyzes development (and vice versa) [6]

How can using a VCS improve the CS 1653 project for our students?

Stay organized: students review their changes when committing

Commit logs improve communication: see what your groupmates have completed

Much simpler merging when working simultaneously: no more emailing code and manually combining!



Continuous submission: work until the deadline, committing as you go

What about using analytics?

VCS analytics to improve the course project

High-level goal: improve group performance… how?

During the semester

- Use analytics to detect problems in groups
- Allow the instructor to intervene as needed

Between semesters

- Use analytics to discover what makes some groups more successful
- Adjust course to encourage behavior seen in strong groups

We collected data from Spring 2015 offering of the course, and applied the lessons learned to Spring 2016

- 2154: 33 students, 14 groups, 4 project phases
- 2164: 33 students, 12 groups, 5 project phases

What analytics correlated with group performance?

Distribution of labor

Author	Rows	Stability	Age	% in comments
	887	81.4	2.3	4.06
	864	71.7	2.6	13.31
	702	58.7	1.4	11.82

Per-week work completed

Author	2015W05	2015W06	2015W07	2015W09	2015W10
Modified Rows:	304	1222	396	1101	1414

Others: Good commit messages, working on documentation early

For Spring 2016, our changes were primarily interventionary

During office hours meetings, checked logs for indicators:

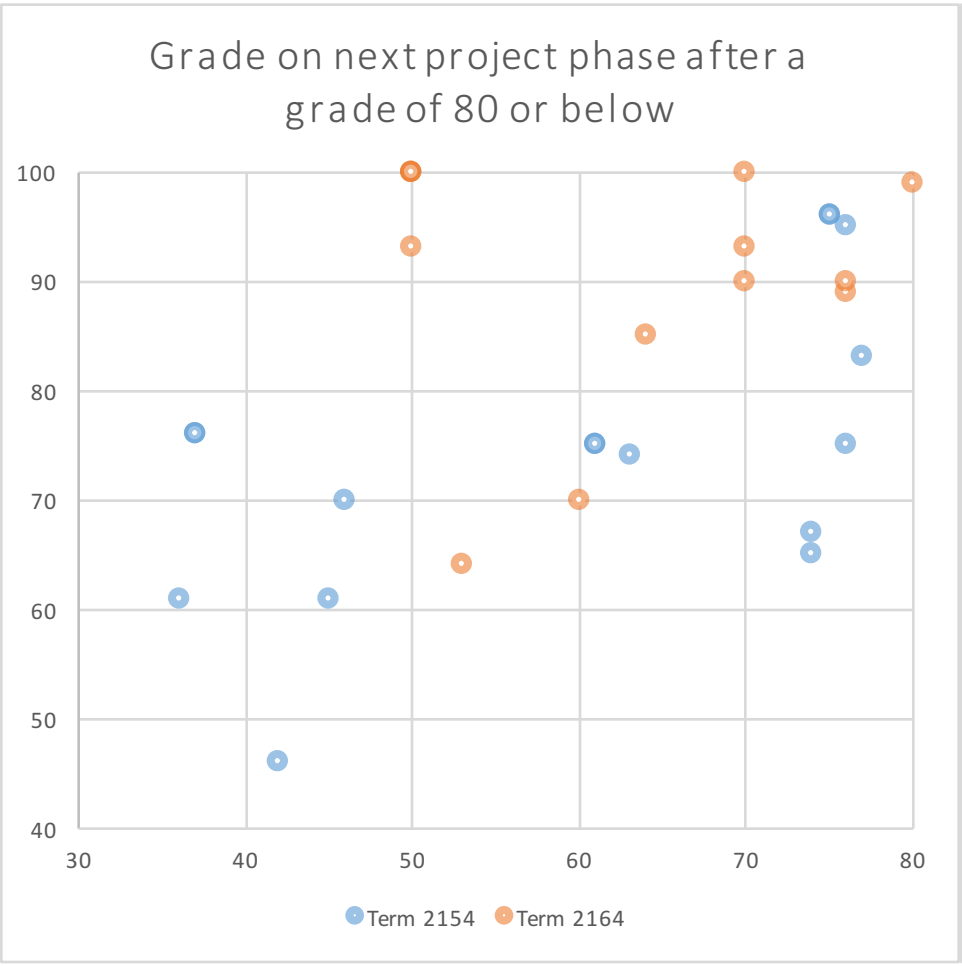
- Early work on documentation
- Balance of commits per member
- Descriptive commit messages

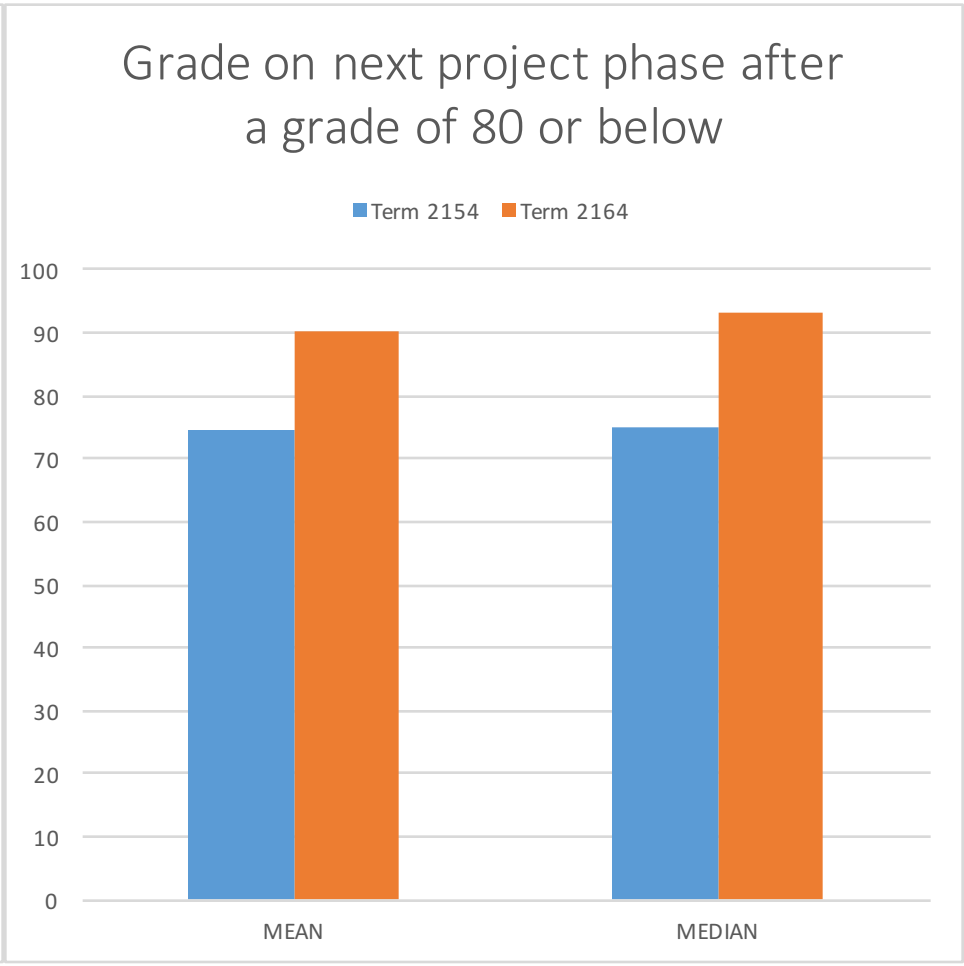
When confronted with concerns, students had a range of responses:

- Expressed regret, admitted they needed to improve
- Defended their groupmates
- Explained special circumstances
  - “John couldn't commit, so changes went through me”
  - “We met at my place and pair-programmed”

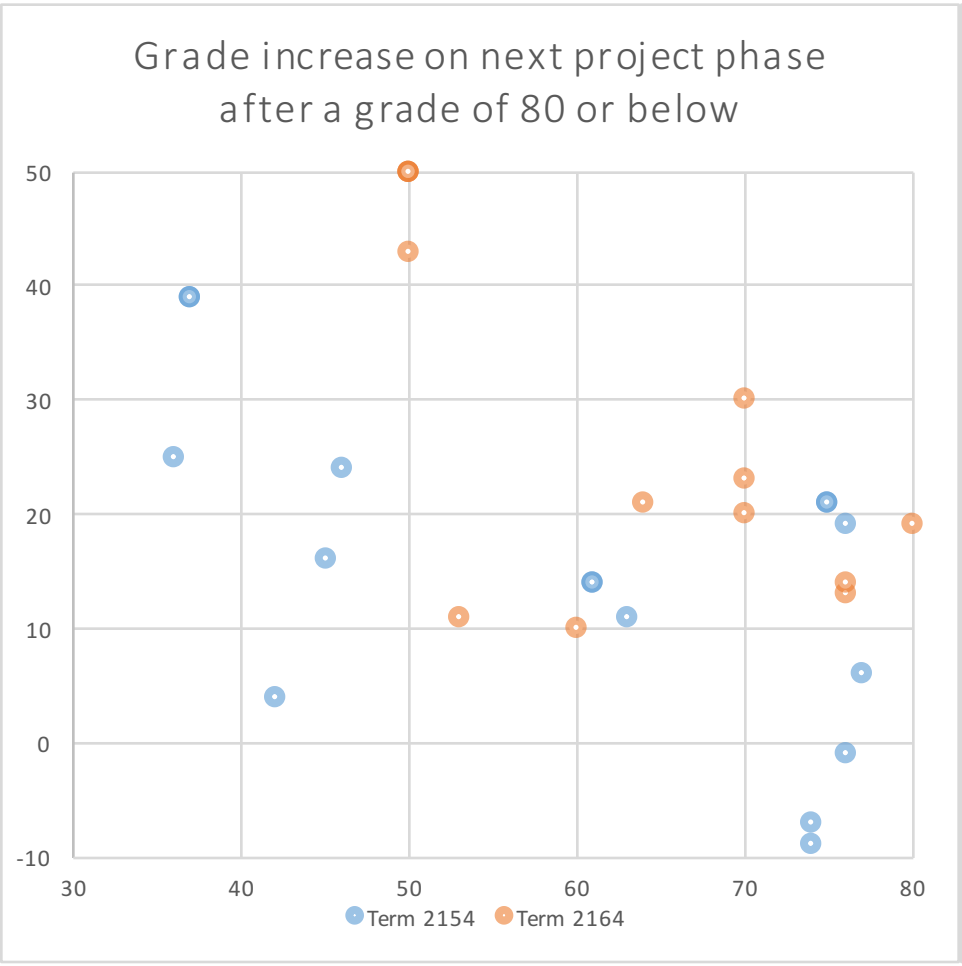
In between phases, offered help managing group work, etc.

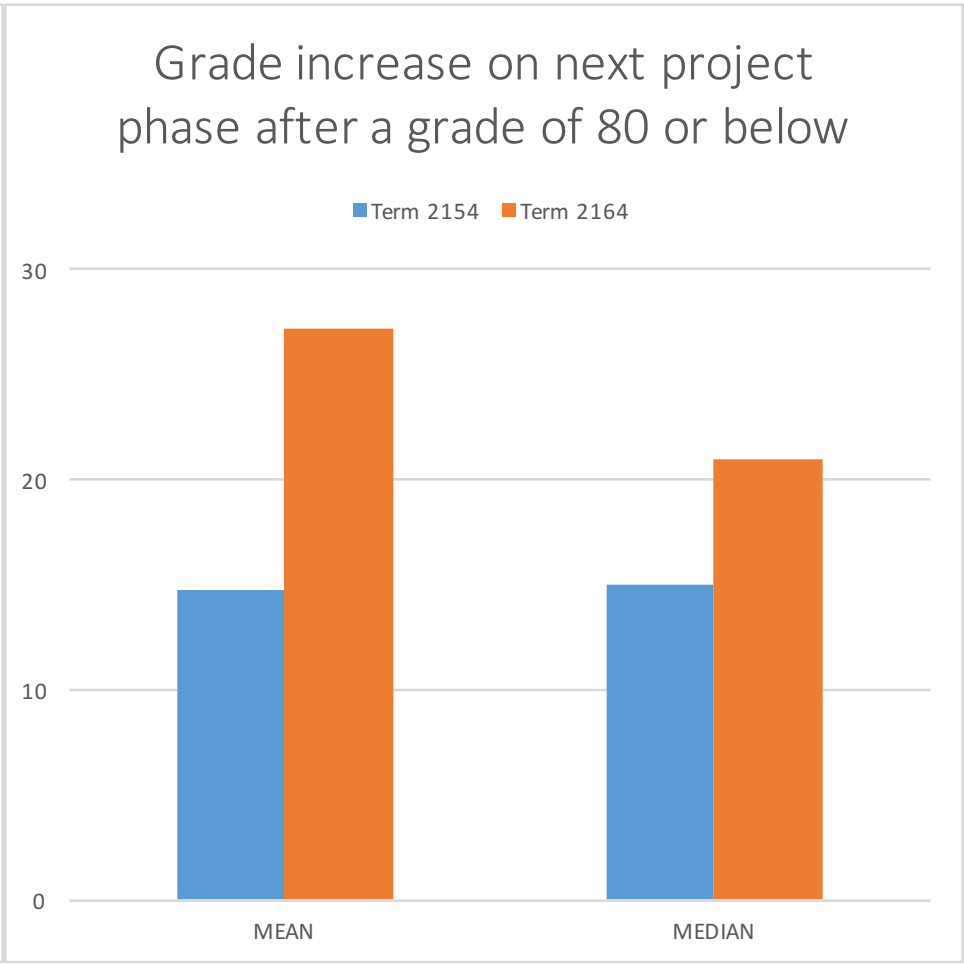
Overall, students seemed to "bounce back" more successfully



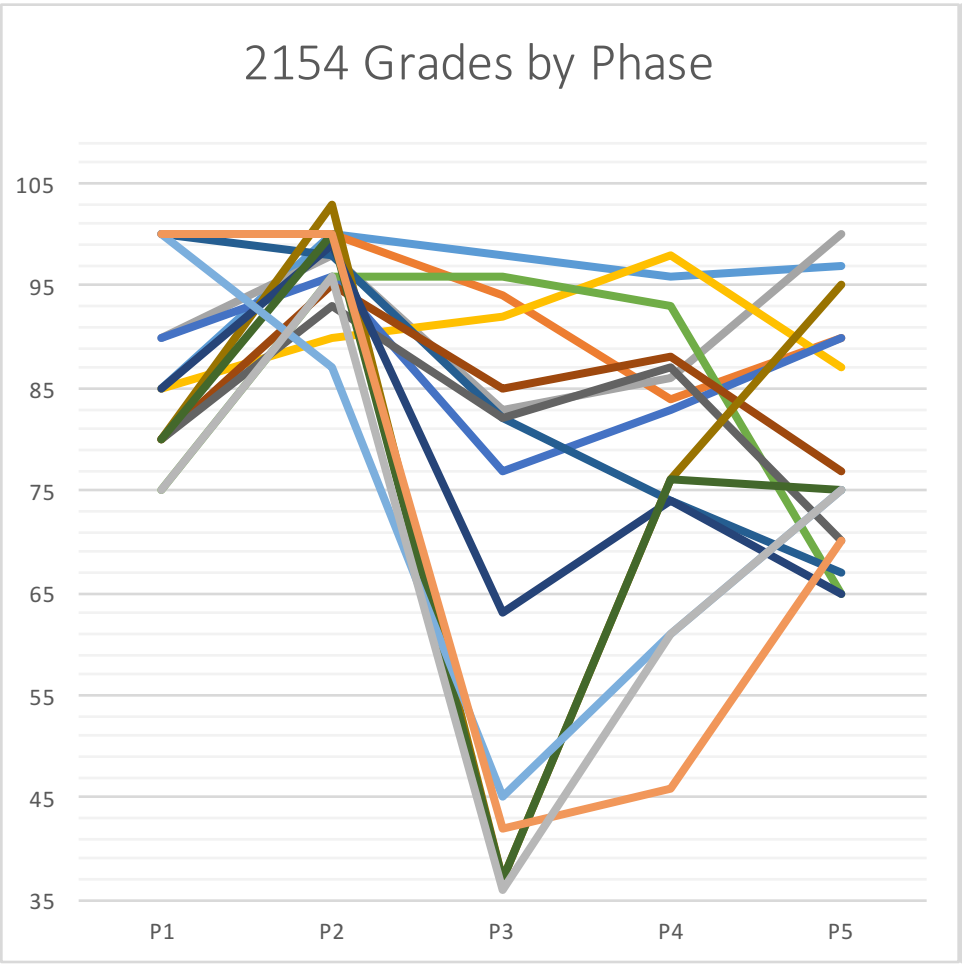


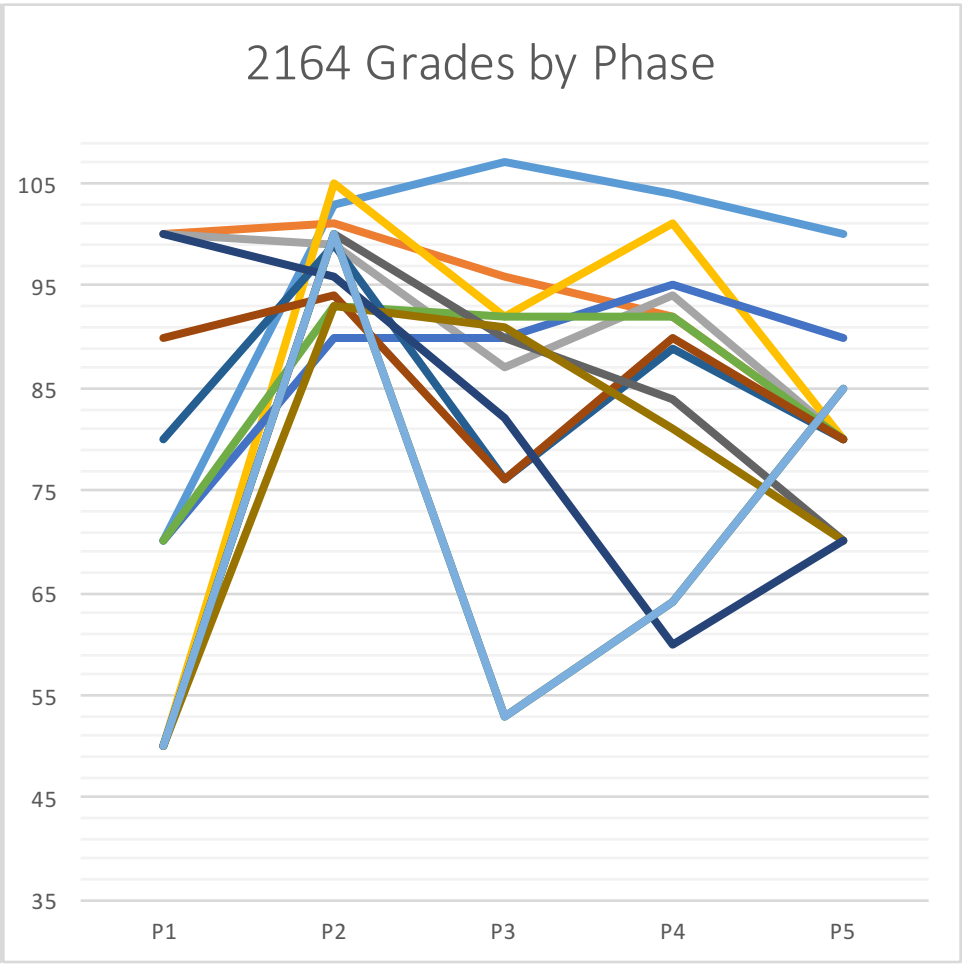
Overall, students seemed to “bounce back” more successfully





Project grades over the course of the term





Future improvements to be made using these techniques

Phase 1 seemed to be harder due to version control

- Hold off until Phase 2?
- Shorter assignment to get used to VC?
- Grade more leniently?

Phase 3 is still the hardest overall

- Closer tracking of repositories, even outside of meetings?
- Give more guidance, leave later phases more open-ended?
- Shorten, move some material to Phase 4?

Outlier groups never recover

- Offer more pointed guidance?
- Detect this type of group, break up early?

Questions?

Thank you!

References:

1. Ross J. Anderson and Roger M. Needham, "Programming Satan's Computer," In *Computer Science Today: Recent Trends and Developments*, 1995.
2. Tyler McDonnell, Baishakhi Ray, Miryung Kim: An Empirical Study of API Stability and Adoption in the Android Ecosystem. ICSM 2013:70-79
3. Baishakhi Ray, Daryl Posnett, Vladimir Filkov, Premkumar T. Devanbu: A large scale study of programming languages and code quality in github. SIGSOFT FSE 2014:155-165
4. Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark G. J. van den Brand, Alexander Serebrenik, Premkumar T. Devanbu, Vladimir Filkov: Gender and Tenure Diversity in GitHub Teams. CHI 2015:3789-3798
5. Casalnuovo Casey, Devanbu Prem, Oliveira Abilio, Filkov Vladimir, and Baishakhi Ray: Assert Use in GitHub Projects. ICSE 2015
6. Bogdan Vasilescu, Vladimir Filkov, Alexander Serebrenik: StackOverflow and GitHub: Associations between Software Development and Crowdsourced Knowledge. SocialCom 2013:188-195