

Programmer-based Fault Prediction

Elaine Weyuker

Thomas Ostrand

AT&T Labs – Research, Florham Park, NJ, USA

PROMISE 2010 Timisoara, Romania

Sep 2010

Defect prediction for large legacy systems

- Negative binomial regression models, based on a simple set of independent variables, are able to predict the files most likely to have faults in the next release of large systems.
- We have applied these models to 7 large systems, with consistently good results

Systems We've Studied

System	Life Span	Releases	LOC in last release	# Files in last release	Average % files with faults	Average % of faulty code
Inventory	4 years	17	538,000	1950	12%	36%
Provisioning 1	2 years	9	438,000	2271	1.3%	5.7%
Voice Response	2.25 years	9	329,000	1926	10.1%	16.9%
Business Maintenance A	9+ years	35	442,000	668	4.8%	17.0%
Business Maintenance B	9+ years	35	383,700	1413	1.9%	13.7%
Business Maintenance C	7 years	27	327,400	584	4.8%	14.3%
Provisioning 2	4 years	16	945,000	3085	6.6%	22.2%

Prediction Results

Actual faults found in predicted top 20% of files
(average over period studied)

System	Life Span	Releases	LOC in last release	# Files in last release	Average % defects in files predicted to be most faulty
Inventory	4 years	17	538,000	1950	83%
Provisioning 1	2 years	9	438,000	2271	83%
Voice Response	2.25 years	9	329,000	1926	75%
Business Maintenance A	9+ years	35	442,000	668	83%
Business Maintenance B	9+ years	35	383,700	1413	94%
Business Maintenance C	7 years	27	327,400	584	85%
Provisioning 2	4 years	16	945,000	3085	87%

The Standard Model

- Predictor Variables
 - KLOC
 - Previous faults (n-1, n-2)
 - Previous changes (n-1, n-2)
 - File age (number of releases)
 - File status (new, changed, unchanged)
 - File type (C,C++,java,sql,make,sh,perl,...)
- Underlying statistical model
 - Negative binomial

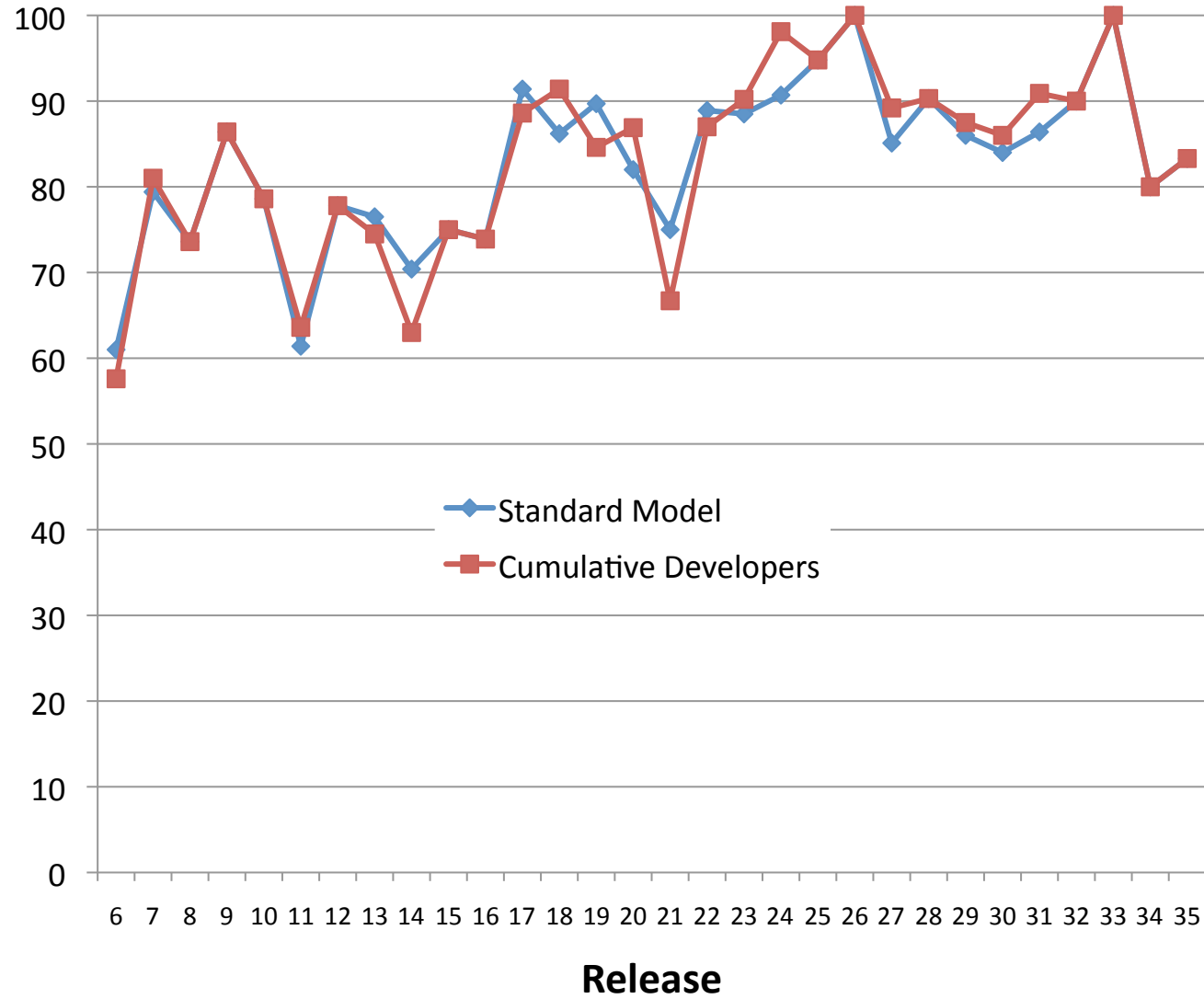
Developer variables

How many different people have modified the code recently?

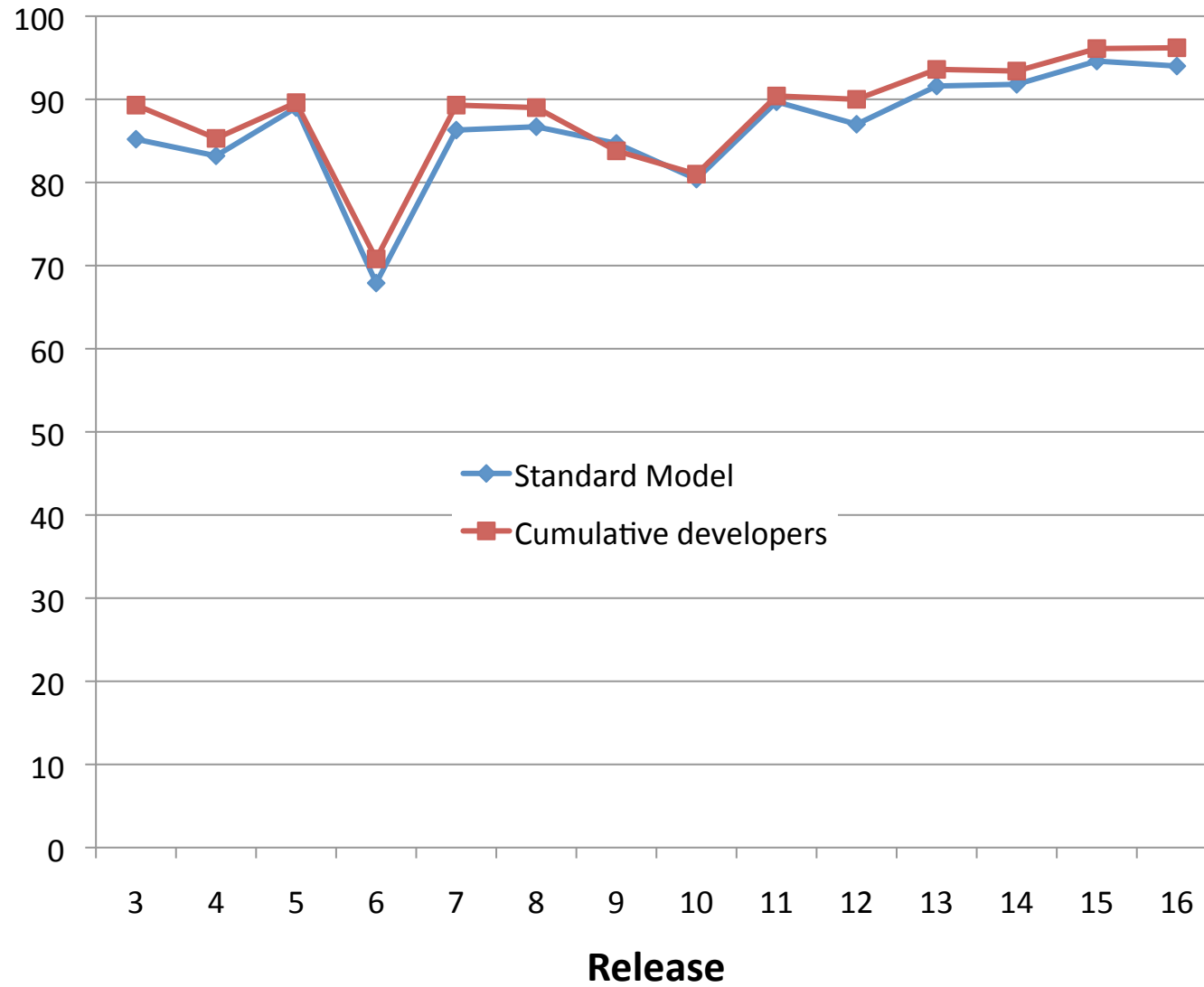
How many different people have ever modified the code?

How many people modified the code for the first time?

Faults in top 20% of files, Maintenance A system

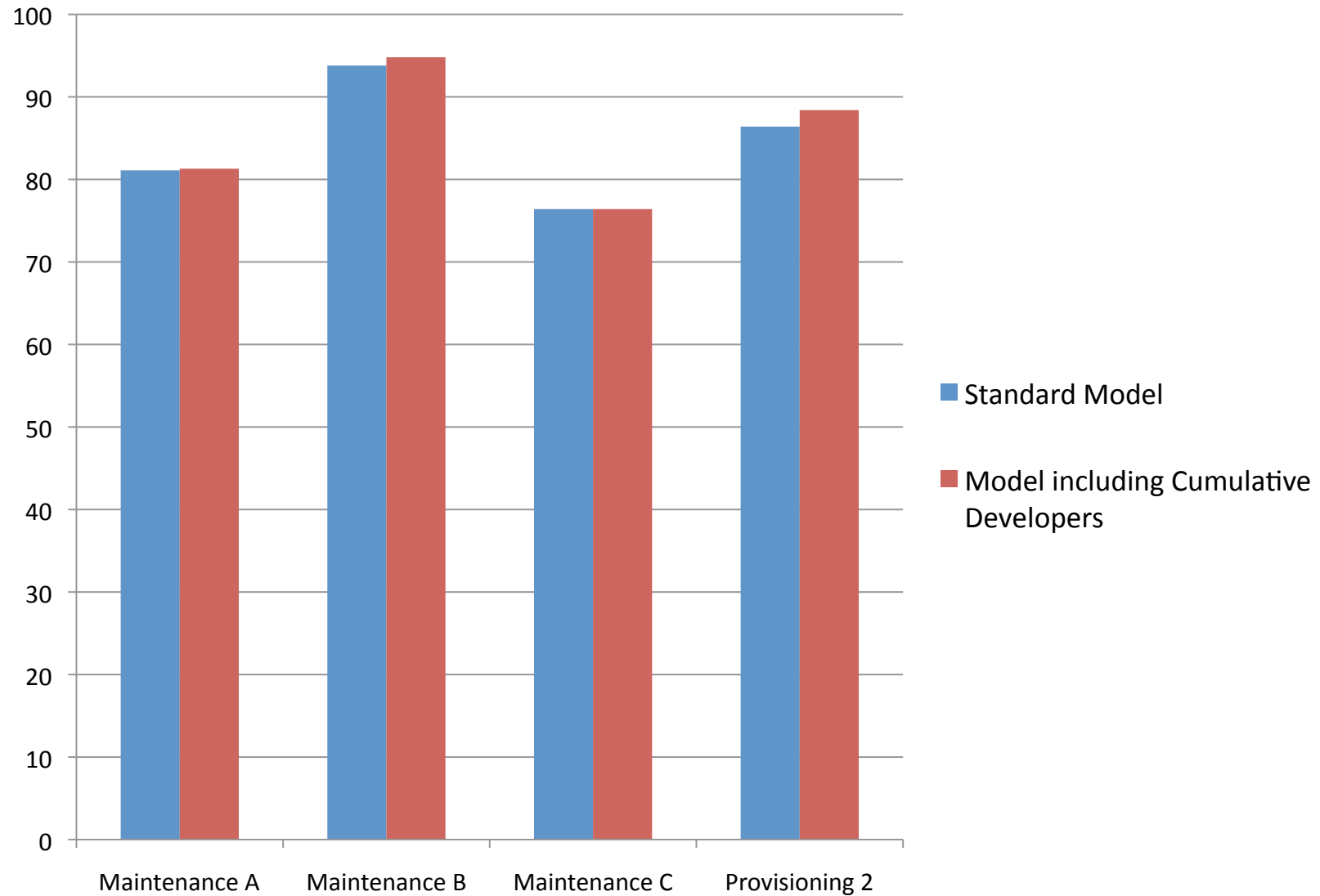


Faults in top 20% of files, Provisioning 2 system



Faults in top 20%, Standard and Cumulative Developer models

Averages over all releases



Developer variables

How many different people have modified the code recently?

How many different people have ever modified the code?

How many people modified the code for the first time?

Which individual worked on the code?

Individual Developers

Previous results are based on all developers who worked on a file.

Suppose we know that a particular developer worked on file F in release N .

What can that tell us about probable faults in F in release $N+1$?

Individual Developers

How can we measure the effect that a single developer has on the faultiness of a file?

If developer d modifies k files in release N

- how many of those files have bugs in release $N+1$?
- how many bugs are in those files in release $N+1$?

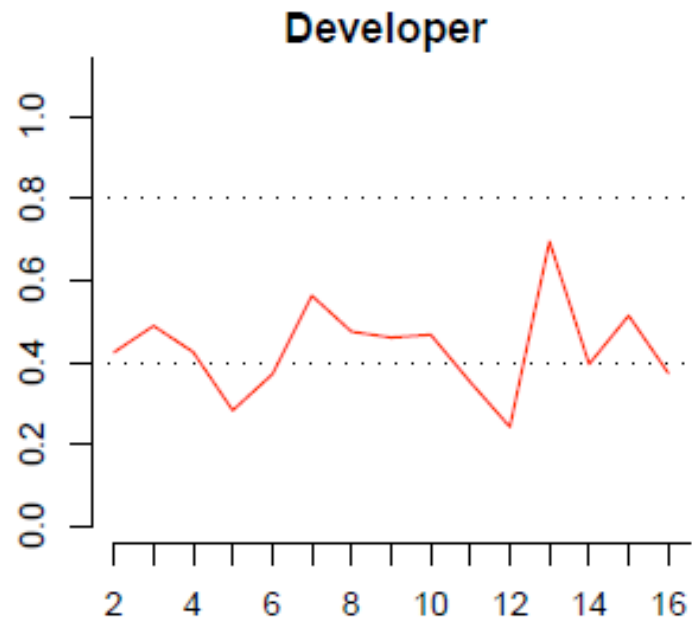
The BuggyFile Ratio

If d modifies k files in release N , and if b of them have bugs in release $N+1$, the *buggyfile ratio* for d is b/k

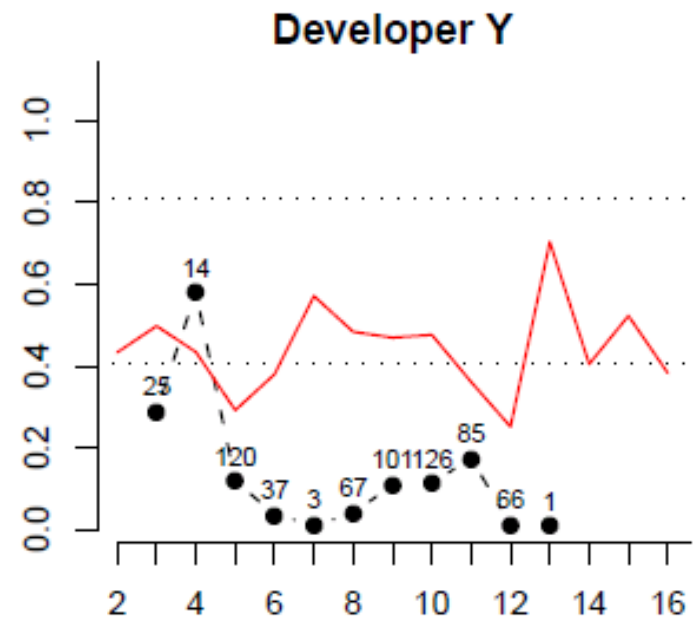
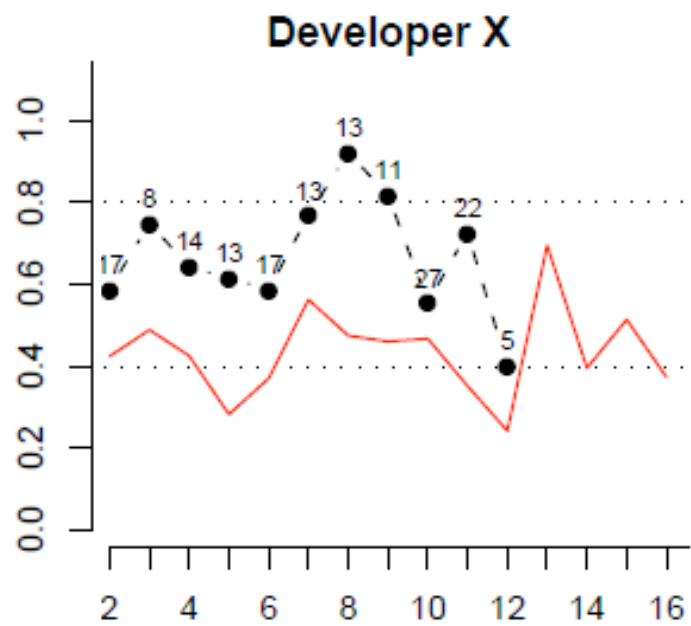
Provisioning 2 has 107 programmers.

Over 15 releases, their buggyfile ratios vary between 0 and 1

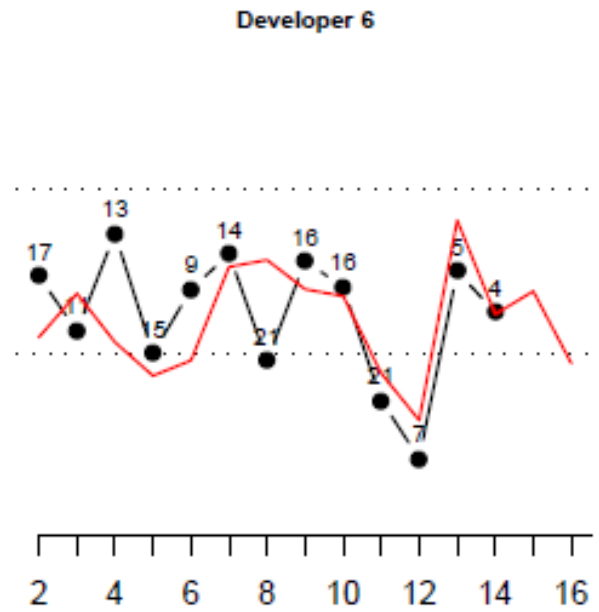
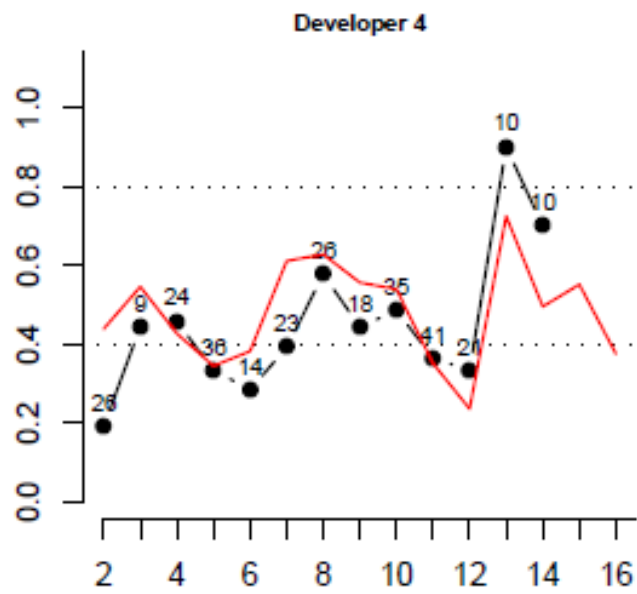
The average is about 0.4



Average buggyfile ratio, all programmers



Buggyfile ratio



Buggyfile ratio

The Bug Ratio

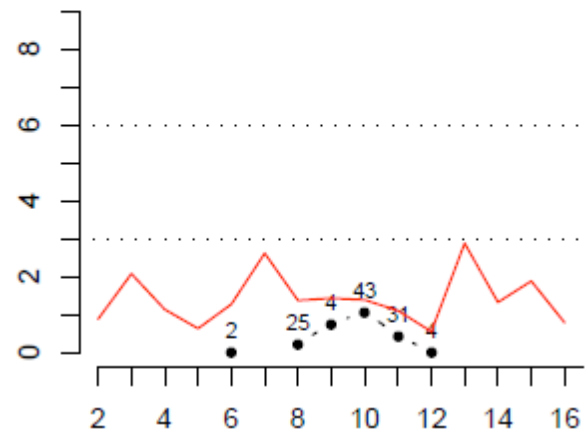
If d modifies k files in release N , and if there are B bugs in those files in release $N+1$, the *bug ratio* for d is B/k

The bug ratio can vary between 0 and B

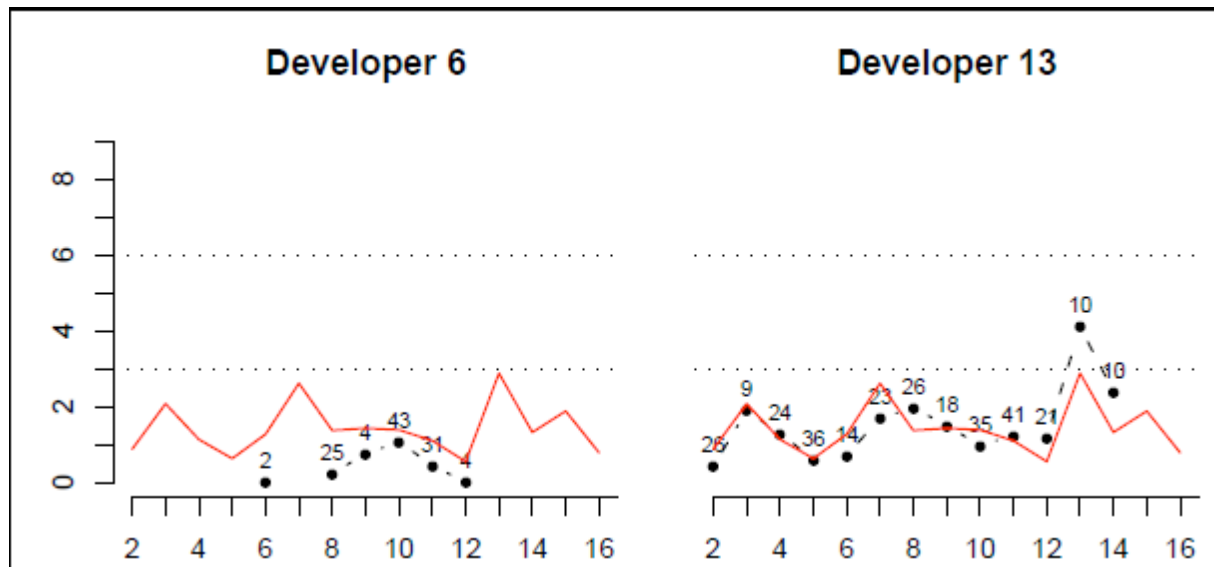
Over 15 releases, we've seen a maximum bug ratio of about 8

The average is about 1.5

Developer 6

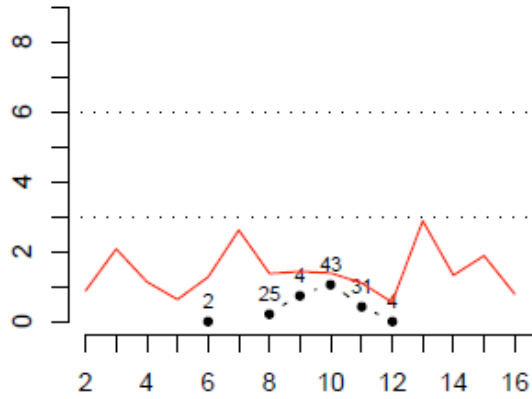


Bug ratio

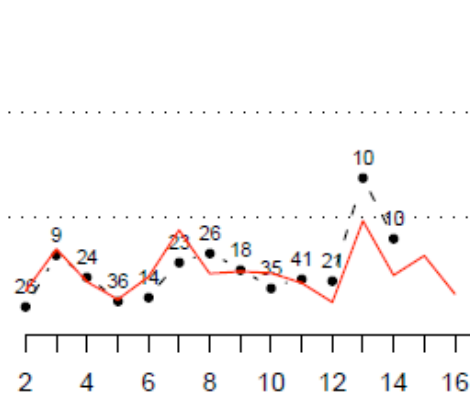


Bug ratio

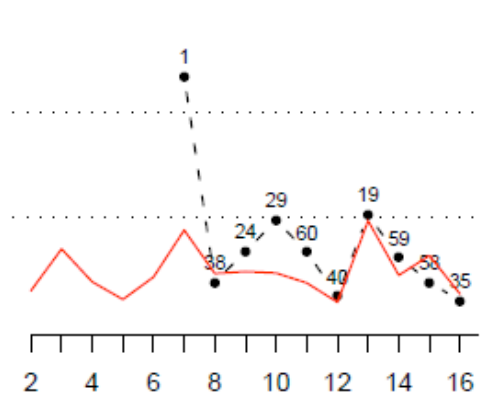
Developer 6



Developer 13

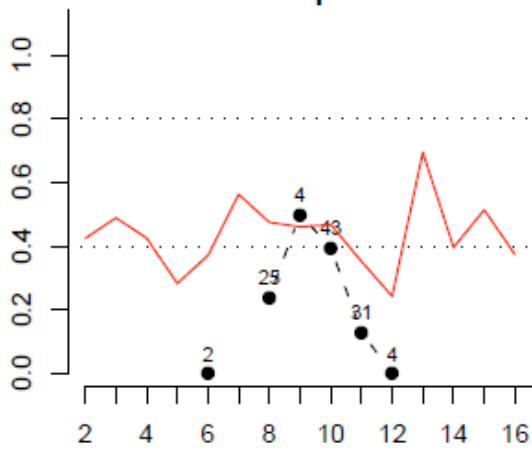


Developer 31

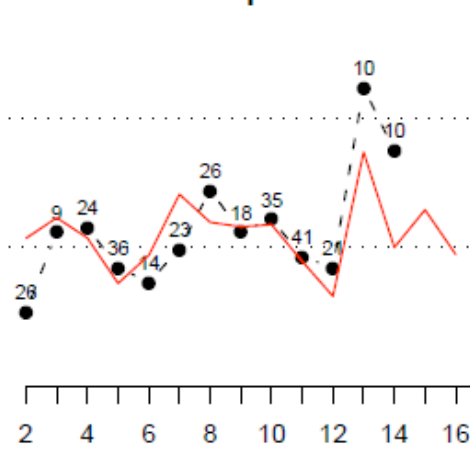


Bug Ratio

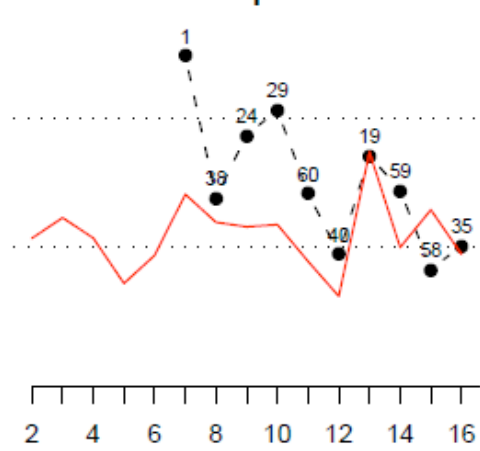
Developer 6



Developer 13



Developer 31



Buggyfile Ratio

Problems with these definitions

A file can be changed by more than one developer.

Just because a file was changed in Rel N and fault was detected in N+1, doesn't mean that the fault was caused by making that change.

A programmer might change many files in the identical way (interface, variable name, ...)

The "best" programmers might be assigned to work on the most difficult files.

For most programmers, the bug ratios vary widely from release to release.

Possible approaches

- Developer factors
 - the number of times developer D touches the file
 - the size of change made by developer D
 - developer clusters
- File factors
 - the file's length
 - the file's previous bug history
 - the number of different developers who touch the file

Future Work

- analyze bug ratio and buggyfile ratio on additional systems
- define and study finer-grained versions of bug ratio and buggyfile ratio

Questions?