Reflections and Perspectives on Predictive Modeling in Industry

Tom Ostrand
AT&T Labs - Research

Crest Open Workshop
Oct 24-25, 2011

Overview

- Fault prediction at AT&T

- Fault prediction and SBSE

AT&T software production environment

- many large, long-lived (legacy) systems
- telecom applications, internal bookkeeping systems, consumer & business customer-facing applications
- multi-(file, module) systems
- more or less regular release schedules (with exceptions)
- most programming in C, C++ or Java
- many other languages, including some special purpose proprietary

Prediction Research Goals

- Develop prediction models that identify parts of system that are most likely to be fault-prone in future releases.
- Provide results to developers, testers & managers, to use in
  - code design & refactoring
  - testing strategy & tactics
  - resource allocation
  - scheduling decisions

Prediction Research Goals

- Discover and apply the best possible model to predict faulty files in next release
- Realize that different systems may have different best models.
- Although the basic shape of the model may be similar across systems, there may be individual adaptations needed.
- Find a "good-enough" model to make predictions, apply it to a wide variety of systems, make necessary adaptations, evaluate each one, and present results to the consumers (system developers and testers)

Fault Prediction Research at AT&T Labs

- 2002-03: Contacting development groups, collecting data, looking for patterns, analyzing relations
- 2003-04: Building & evaluating first prediction models, creating the *Standard Model*
- 2005-07: Applying models to more systems, evaluating model variations
- 2007-09: Evaluating different modeling approaches, and more model variations
- 2009: Construction of user-oriented tool
- 2010-11: Integrating finer-grained data into model

Systems we've studied

System	Years followed	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%
Utility	4.5 years	18	281,000	802	5.9%	25.7%
Inventory 2	4.5 years	18	2,115,000	6693	2.9%	18.5%

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	76%
Provisioning 2	4 years	16	945,000	3085	87%
Utility	4.5 years	18	281,000	802	86.5%
Inventory 2	4.5 years	18	2,115,000	6693	92.5%

Observations

- Doing this kind of research takes a long time
- It’s crucial to establish good relations with the people in the production groups
- Trust is important
- Look for individual(s) in the product group who are enthusiastic about the technology
- There are many similarities between projects
- No two projects are the same

Difficulties

- Which data are reliable?
- Which change requests are bugs?
- What to do about unreliable data?
- Persuading projects to try the technology
- Persuading projects to adopt the technology

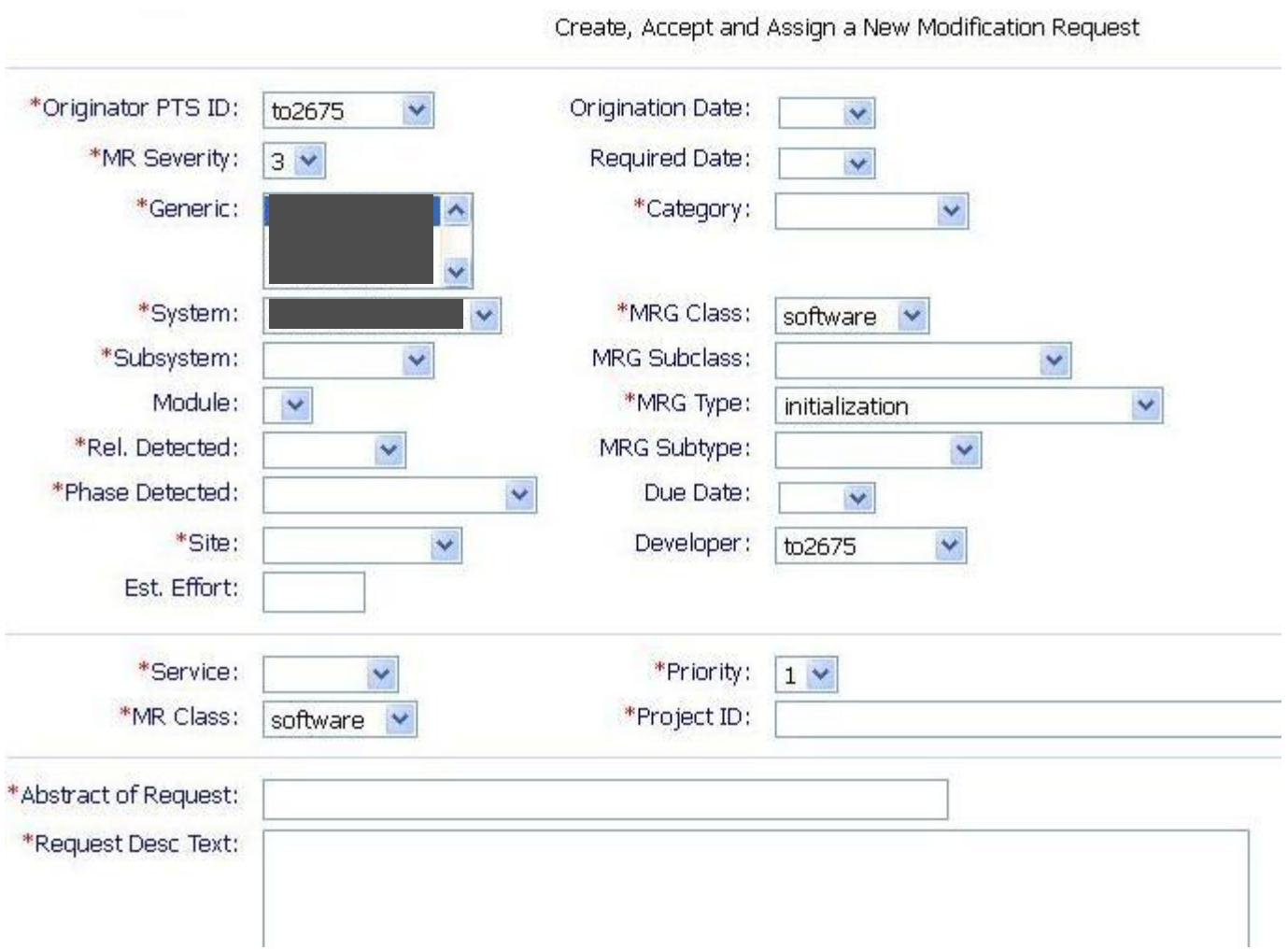
Which data are reliable?

- Change request data are collected both automatically and from manual input
- Automatic collection is *usually* reliable
  - system clock incorrect (DST)
  - check-out/check-in a file without changing it can lead to spurious change counts.
- Manual input
  - Default values
  - Careless choice
  - Fields used for different purpose
  - Deliberate misinformation

Which change requests are bugs?

- Techniques we have considered
  - Category of the MR
  - Role of the person filing the change request
  - Project phase when the change request is filed
  - Text analysis of natural lang description
- Problems
  - Category field doesn't exist in the MR
  - Category field values are misleading or unclear
  - Fields are used differently by different projects
  - Users choose wrong value or leave default for category

MR creation form



MR fields used by a project

Category

- Action
- Issue
- Enhancement
- Modification
- Defect
- Other

Type

- Initialization
- New feature
- Change to existing feature
- Bug fix
- New work request
- Other

How do we handle unreliable data?

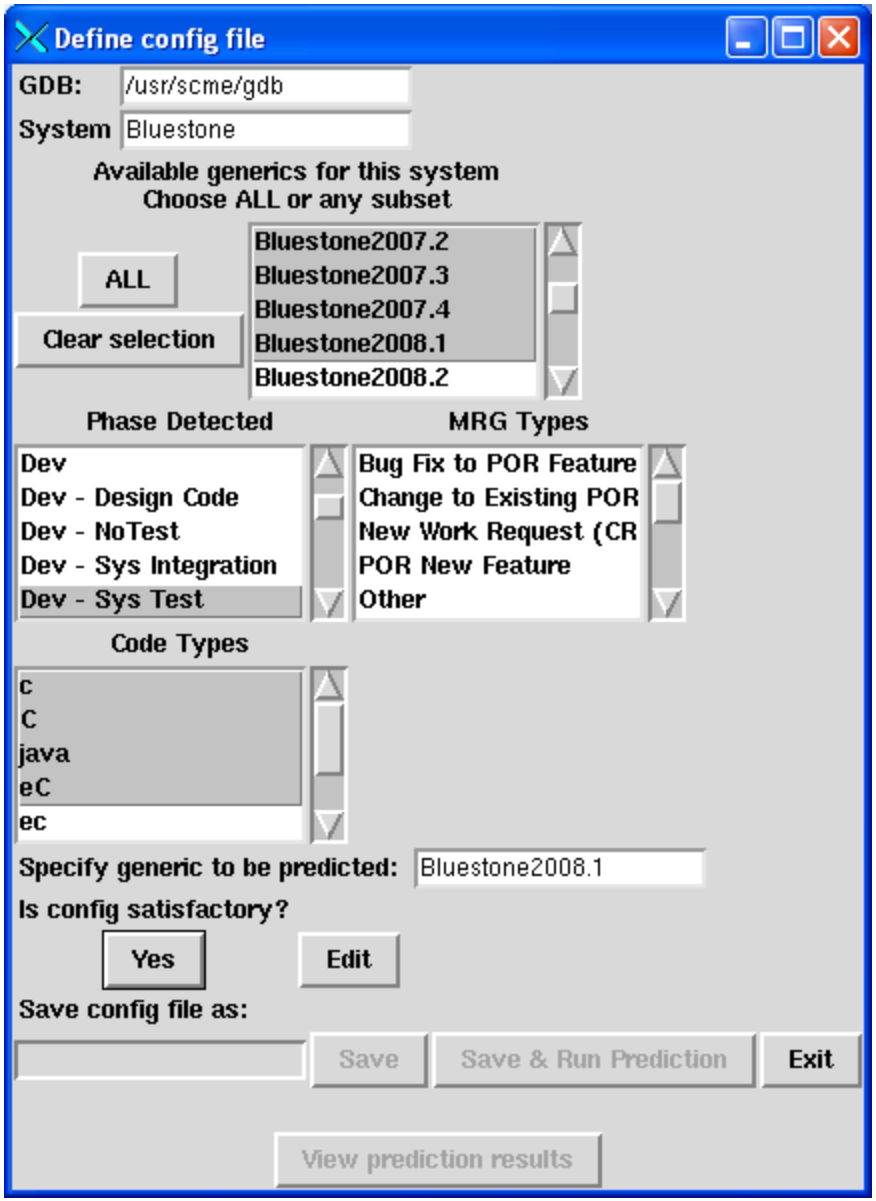
- Don't use it
- Find substitute data to convey the desired information
- Convince user group and tool provider to modify data collection procedure

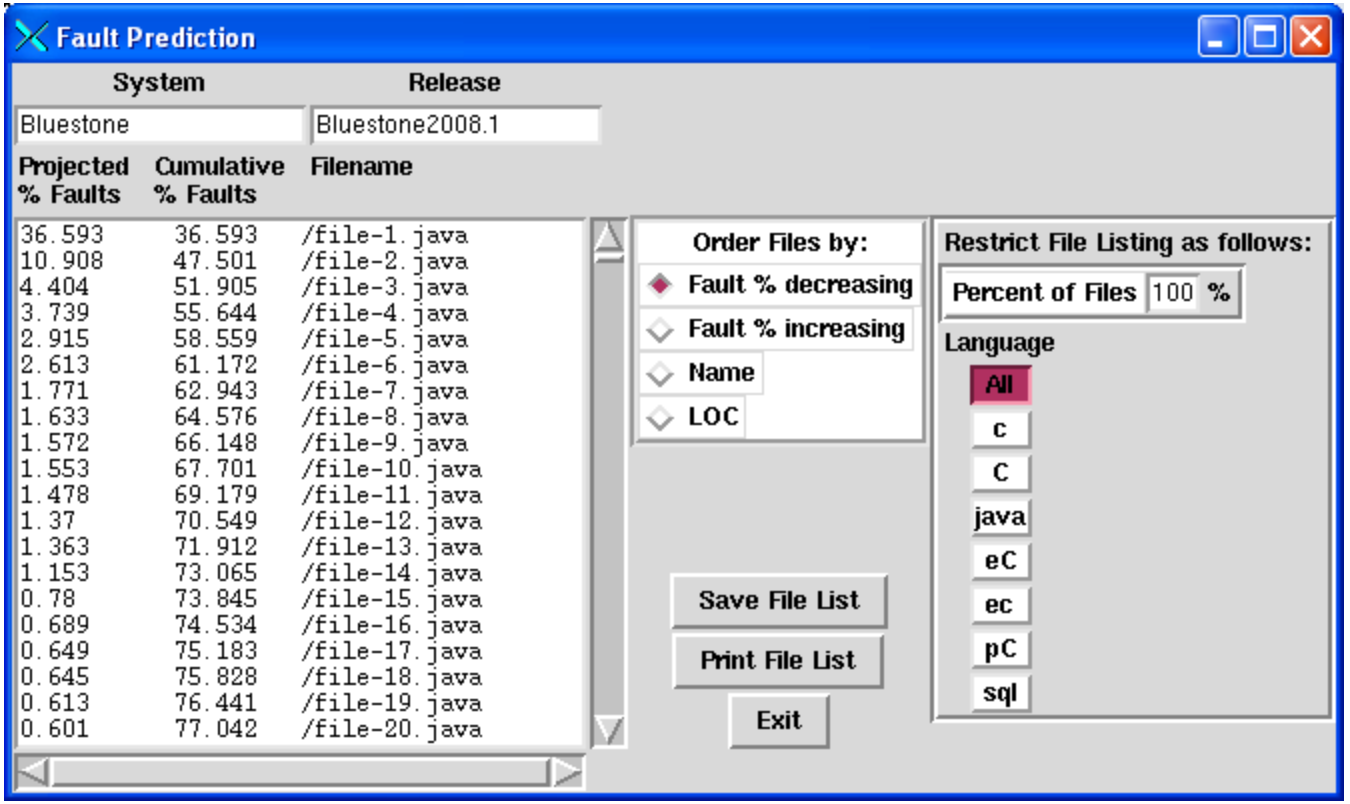
Unreliable data

- Don't use it -- *severity*
- Find substitute data – *use other ways to determine if it's a bug*
- Modify data collection procedure -- *add a new "bug identification" field*

Lessons Learned

- Technology must provide an obvious benefit
- Demonstrate results on target users' data
- Relate the technology to problems of the users
- Tools should be simple, simple, simple
  - little or no training
  - minimal inputs from user; automated data acquisition
  - results easy to interpret


Define config file
GDB:
/usr/scme/gdb
System
Bluestone
Available generics for this system
Choose ALL or any subset
ALL
Clear selection
Bluestone2007.2
Bluestone2007.3
Bluestone2007.4
Bluestone2008.1
Bluestone2008.2
Phase Detected
Dev
Dev - Design Code
Dev - NoTest
Dev - Sys Integration
Dev - Sys Test
MRG Types
Bug Fix to POR Feature
Change to Existing POR
New Work Request (CR
POR New Feature
Other
Code Types
c
C
java
eC
ec
Specify generic to be predicted:
Bluestone2008.1
Is config satisfactory?
Yes
Edit
Save config file as:
Save
Save & Run Prediction
Exit
View prediction results



Initial prediction view for Bluestone2008.1

All files are listed in decreasing order of predicted faults

The Art and Craft of Technology Transfer

A Tale in 4 Acts

ACT I

- Time
  - Sometime in the past
- Setting
  - An empty conference room, in a building occupied by ~3000 software designers, developers and testers of a large telecom corporation (Building Z)
  - In the room, a conference table with 12 seats
  - On the table, a projector

Act I, Scene 1

- The Players
  - T. Ostrand, E. Weyuker (members of AT&T Research)
  - Manager W, supervisor of ~30 people, managing an inventory project that the researchers have studied, and for which they have achieved good prediction results
  - Manager X, supervisor of ~100 people, managing a project the researchers would like to investigate, and persuade to adopt the technology
  - Manager Y, supervisor of ~120 people, managing a project the researchers would like to investigate, and persuade to adopt the technology

Act I, Scene 1

The empty conference room, 15 minutes before a scheduled meeting.

- *Enter researchers*
- TO: "This room is really well-hidden; we're lucky we found it. Where's the cable for the projector?"
- *Much looking around for the cable.*
- EW: "They'll be here in 5 minutes!"
- *Desperate call to local tech support*
- *Finally the cable is found in a closet, and plugged in to the researcher's laptop. Several attempts to attach to the research network follow.*
- *The researcher manages to connect to his home network, seconds before ….*

Act I, Scene 2

- *Enter Manager W*
- Mgr W: “I never knew this room was here. It’s very far from my office”
- TO: “Are the others on their way?”
- Mgr W: “X couldn’t make it this morning, but Y will be here”
- *5 minutes pass.*
- *Enter Manager Y*
- Mgr Y: “I never knew this room was here. It’s terribly far from my office.”

ACT I, SCENE 2 (cont.)

- TO: "Thanks for coming. We're here to show you the fault prediction tool that we've successfully applied to W's system. I've logged into that system, and will run the tool directly on the last release"

- *TO brings up the tool, explains its very simple interface, and starts to type in the required inputs to request fault predictions for the last release (2010.1).*
- Mgr Y: "That release has just been sent to the field, and we're starting system test on the next one. Could you get fault predictions for 2010.2?"
- *TO requests fault predictions for 2010.2.*

Act I, Scene 3

(About 1 minute later)

The prediction results appear, showing the first 20 files in the list of predicted most faulty files.

Since this is run on the real system, the real file names are shown.

Mgr Y looks at the list, and takes out his cell phone.

Mgr Y: “Excuse me a moment”

(dials)

Mgr Y: “X, you better come over here”

Act I, Scene 4

(About 10 minutes later)

Enter Manager X

Mgr X: “I never knew this room was here. It’s farther from my office than anything else in this building”

TO: “Thanks for coming. We’re running a demo of our fault prediction tool”

Mgr Y: “Look at the list of files. Some of these are in your subsystem.

Mgr X: "I know that file1,...,file5 are complex and have problems, but I thought file6 and file7 were in good shape. We’ll do code reviews on them”

ACT I, SCENE 4 (cont.)

The demo continues.

Manager X is impressed that the tool could identify the files that he knew were problematic, and gives assurance that his team will look more closely at file6 and file7.

The presentation concludes with all managers agreeing that the tool was impressive, and worthy of consideration.

ACT II

Time:

About 6 months later

Setting:

The researchers' home office.

Enter TO, EW

TO: "We've had a couple of conversations with the managers since our demo 6 months ago, but so far no request to install the tool and provide some training."

ACT III

Time:

6 months later

Setting:

A large auditorium in Building Z, occupied by ~100 software developers & testers from the company

Occasion:

An all-company symposium on current software methods, products, and tools.

ACT III, SCENE 1

Symposium Program

- *Keynote address, by invited speaker from external software company*
- *..... (other presentations)*
- *Software Fault Prediction Tool, by TO & EW*
- *..... (more presentations)*

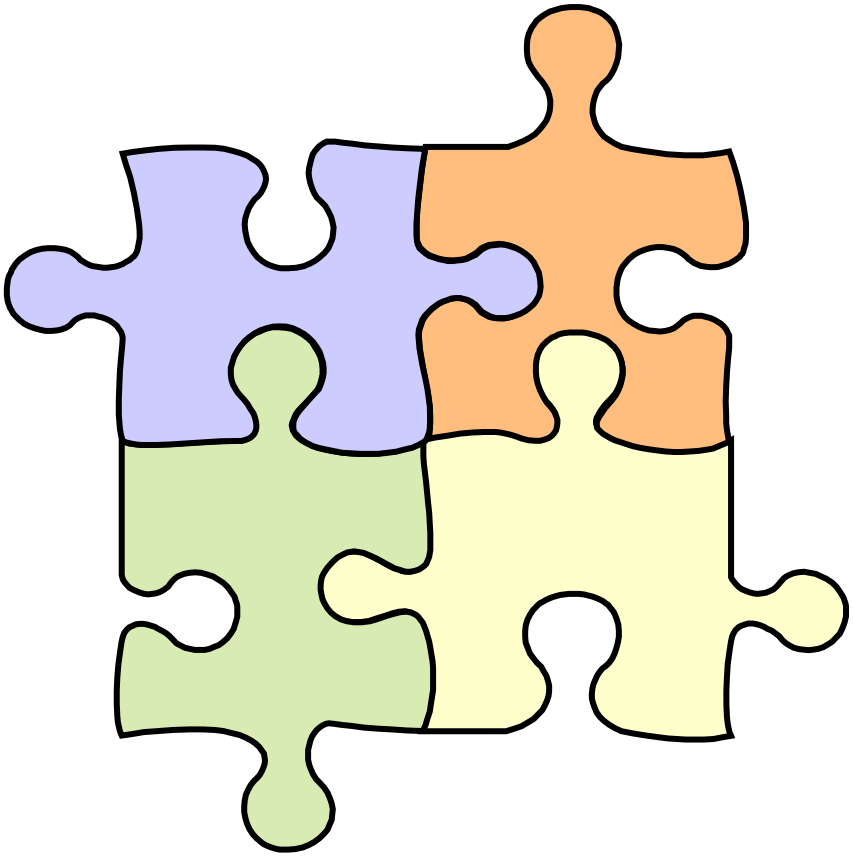
Act III, Scene 2

- *Coffee Break*

Several Symposium attendees express interest in applying the prediction tool to their projects.

They request follow-up information and demos.

Act IV, Sometime in the Future



Fault Prediction and SBSE

- Where have we used SBSE techniques in our prediction work?

- Where could we use SBSE techniques?

Contributions from the audience are welcome and will be gratefully acknowledged.

Search-based Software Engineering

- Problem formulation
- Set of candidate solutions
- Fitness function

Problem Formulation

Given a system with *k* releases, R_1, ..., R_k, R_{k+1} and complete knowledge of faults, changes and file data for releases 1,...,k

Construct a general prediction model that will be made specific for each release, and that yields a predicted fault count for each file of release R_i, using data from releases R_1,...,$R_{(i-1)}$.

That model will then be used to predict fault counts for the files of release R_{k+1}.

Set of candidate solutions

- Data collection procedure
  - Automated extraction from configuration management/version control system
- Attributes that are potential predictor variables
  - static code properties
  - change and fault history
  - developer information
- Format and parameters of a prediction function
  - negative binomial regression

Fitness Function

Meaningful evaluation of prediction quality

- faults in first 20% of files
- fault-percentile average

Brief summary of our prediction work

- Developed the Standard Model based on negative binomial regression.
- Applied Standard Model to 9 large industrial systems, developed both in and out of AT&T
- Examined and evaluated 3 alternative modeling approaches
- Evaluated multiple different sets of independent variables to drive the prediction model
- Defined a new evaluation function to measure the prediction quality

The Standard Model

- Target systems are large (>100,000LOC, 100s of files), long-lived systems with regular periodic releases
- Goal is to identify the files most likely to have faults in the system's next release, before system testing of that release begins
- Output (dependent) variable is *predicted number of faults* per file in the next release of the system

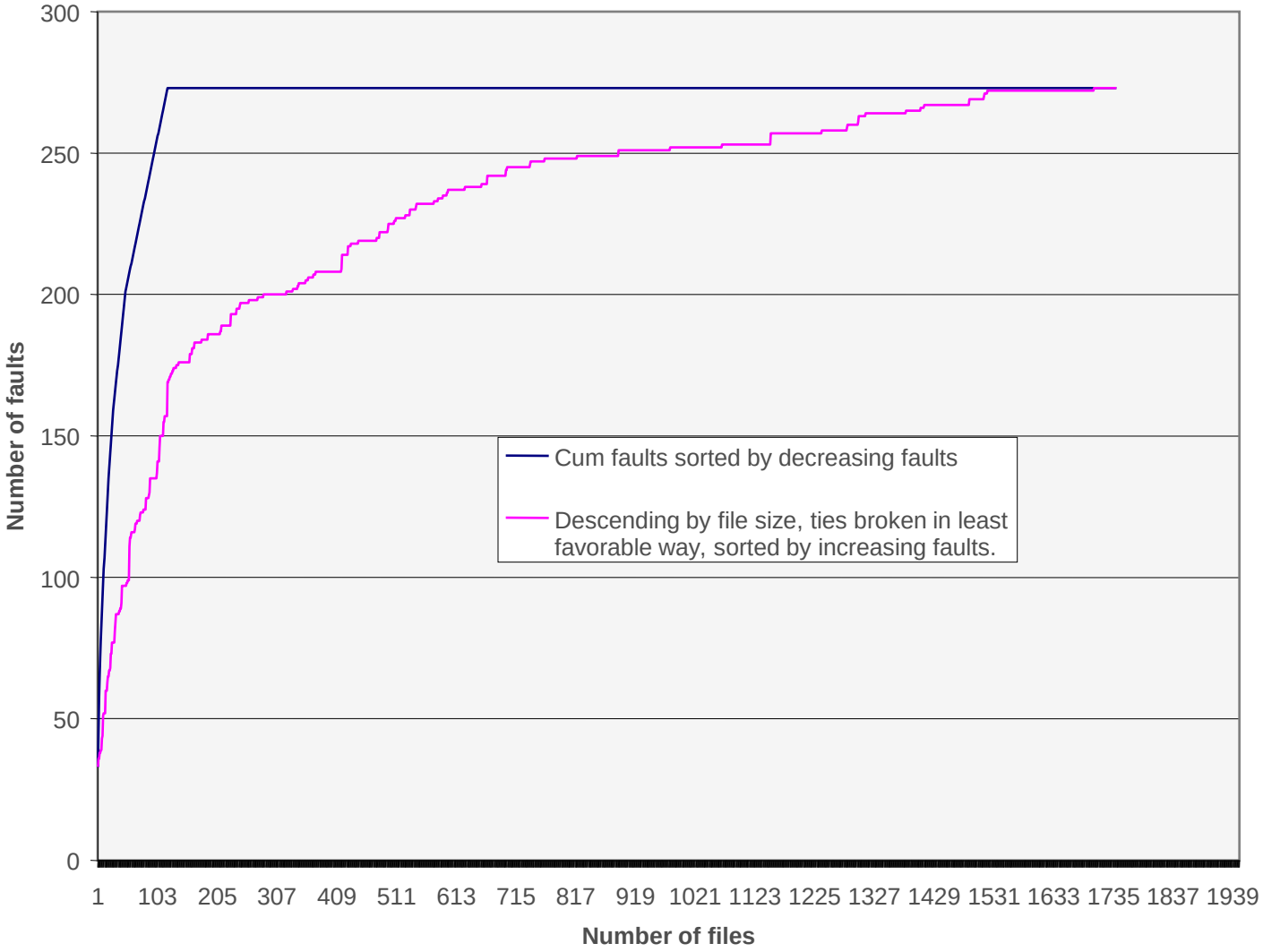
The Standard Model

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

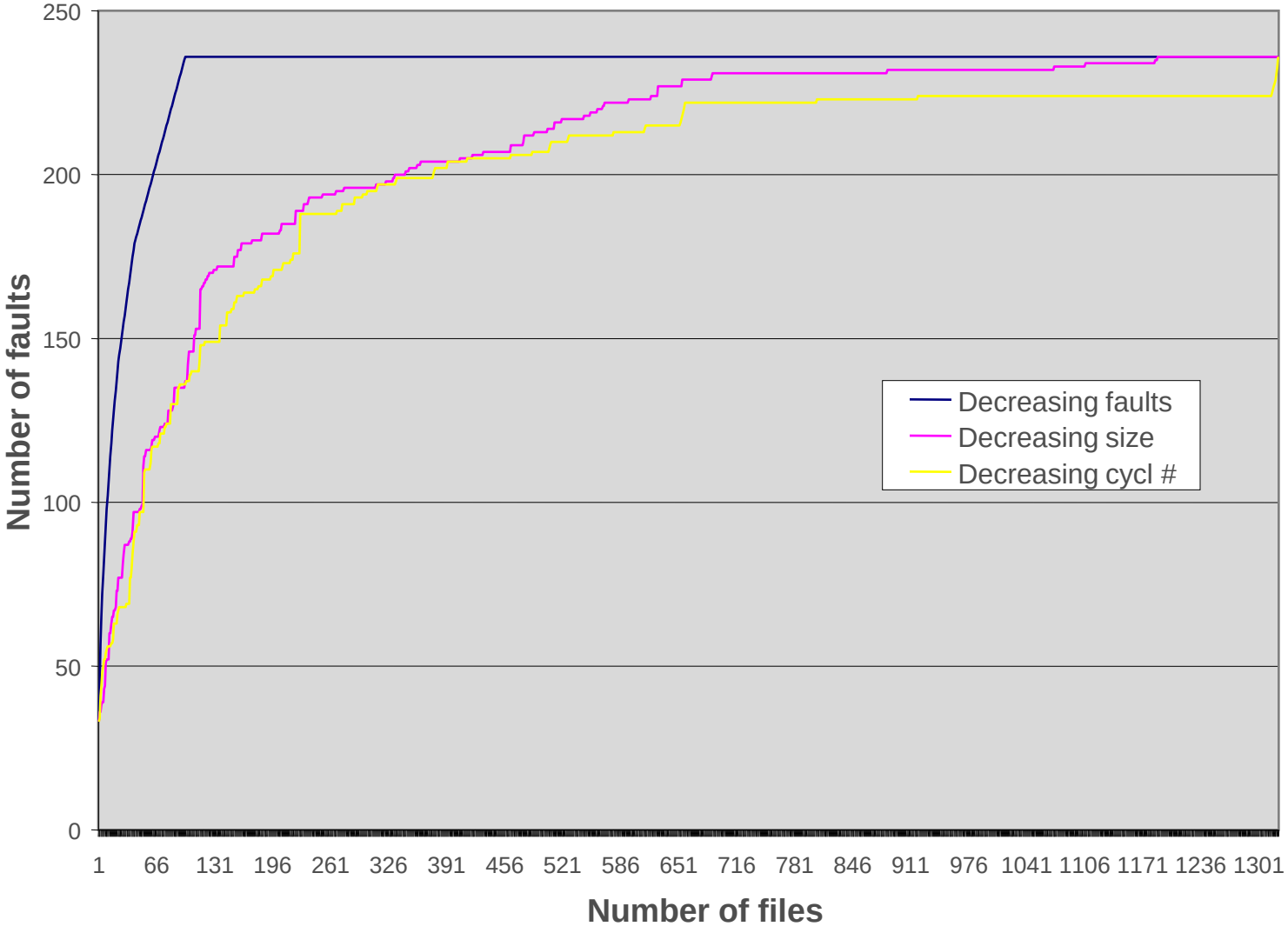
How did we create the Standard Model?

- Search for attributes that are potential predictors
  - Scatter plots of faults vs.
    - LOC
    - counts of previous faults and changes
    - age of file, language
  - Plots of decreasing fault counts
  - Correlations between fault counts and potential predictor variables
- Format of predictor function
  - negative binomial regression
  - examined others (decision tree, random forests, BART)

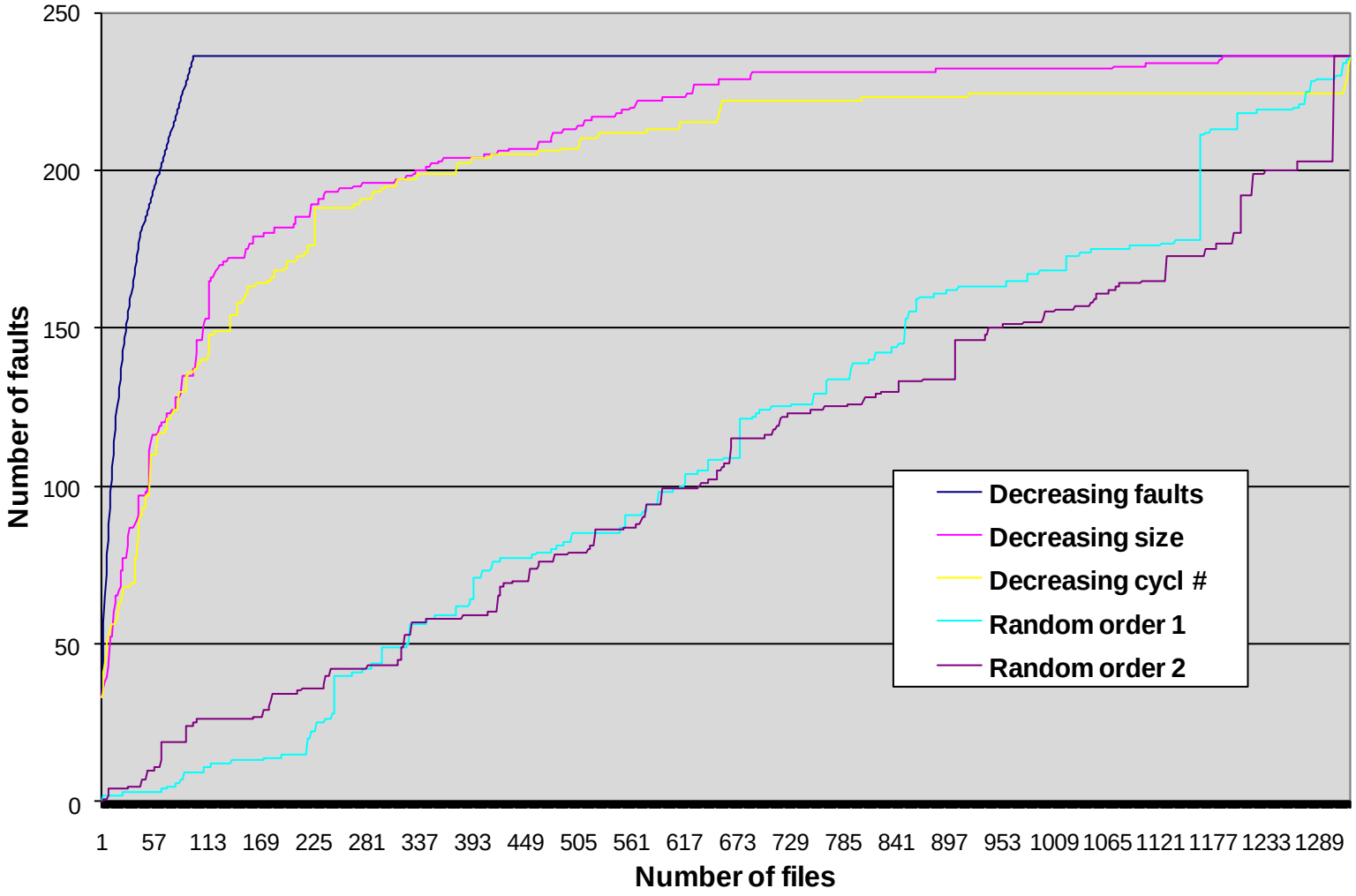
Release 12 Cumulative Faults, by decreasing faults and decreasing file size



Release 12 Cumulative faults, by size and by cyclomatic number



Cumulative faults, by size, cyclomatic number, and random order



Correlations between faults and attributes, Maintenance A

LOC	.231
Changes in N-1	.314
Faults in N-1	.365
New File	.032
Developers in N-1	.303
Developers in 1,...,N-1	.228
Callers (in N)	.084
Callees (in N)	.195
New Callers	.015
New Callees	.028
Faulty Callers in N-1	.184
Faulty Callees in N-1	.278
Changed Callers in N-1	.160
Changed Callees in N-1	.253

Fitness Function

- Our goal is to predict *Fault-Proneness* of files, not Faulty or non-Faulty
- Prediction produces an ordering of files
- Fitness Fcn 1: the percent of actual faults contained in the top X% of the ranking.
- Fitness Fcn 2: *Fault-percentile average* = the average value of FF1 over all values of X.

Objectives for Fault Prediction
(Harman, *PROMISE2010*)

- Predictive quality/ accuracy of predictions
- Cost
- Privacy
- Readability
- Actionable

Objectives

- **Quality:** Yield of faults in top 20% ranges from 75-94%
- **Cost:** With an automated tool, **cost** is near 0
- **Privacy:**
  + Results are available only to project that requests them
  + Standard model does not include any programmer-related information
  + Developer-augmented models use aggregate counts, not individual.
  - A specific file may be associated with one or more specific programmers. Fault predictions may impact those individuals.
- **Readability:** Tool is designed so that prediction results are very straightforward to interpret.

Objectives

Actionable: Clear use of prediction results

- Consistent high fault predictions over several releases can indicate a need to re-design and re-implement a file (refactoring)
- Files at the top of the predicted fault ranking are subjected to increased test coverage and more frequent testing
- The most skilled testers are assigned to design tests for the most fault-prone files
- Test scheduling can be influenced by the distribution of fault predictions: few high predictions vs. many moderate predictions.

Variations of the fault prediction model

- Predictor variables
  - Developer counts
  - Individual developer history
  - Calling structure
  - Fine-grained changes (churn)

Developer attributes we considered

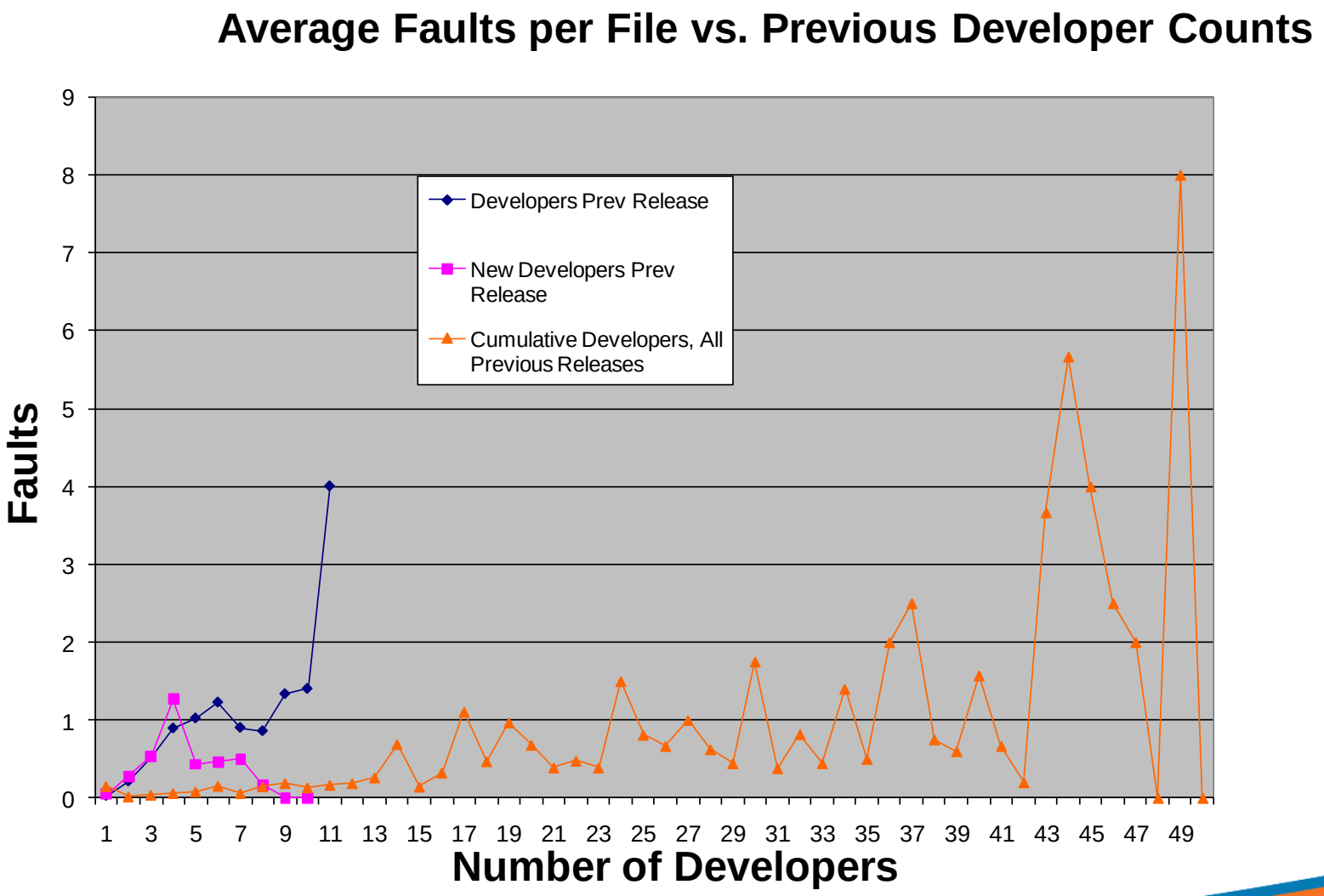
How many different people have worked on the code recently?

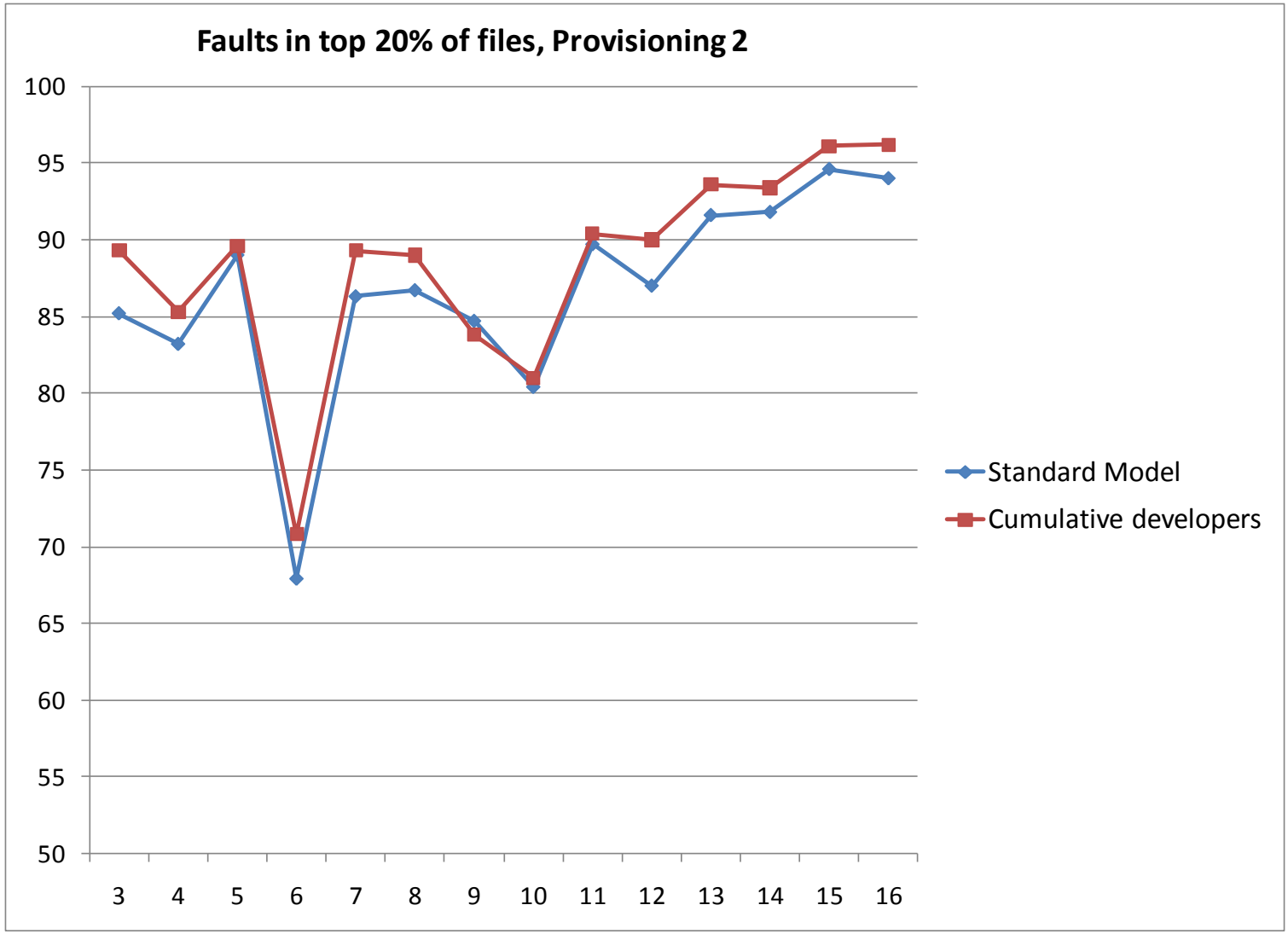
How many different people have ever worked on the code?

How many first-time viewers have changed the code?

Which specific people have worked on the code?

Releases of Business Maintenance A





Total developers touching file in all previous releases

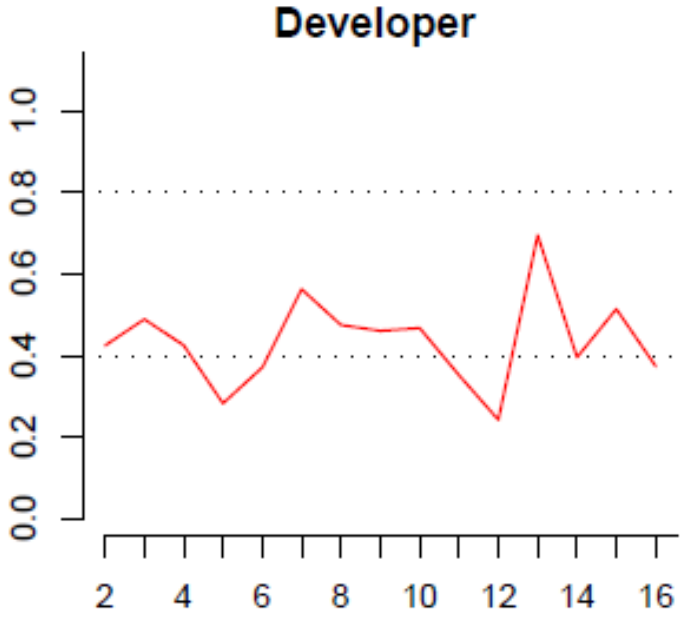
Can we utilize characteristics of individual developers?
The BuggyFile Ratio

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*

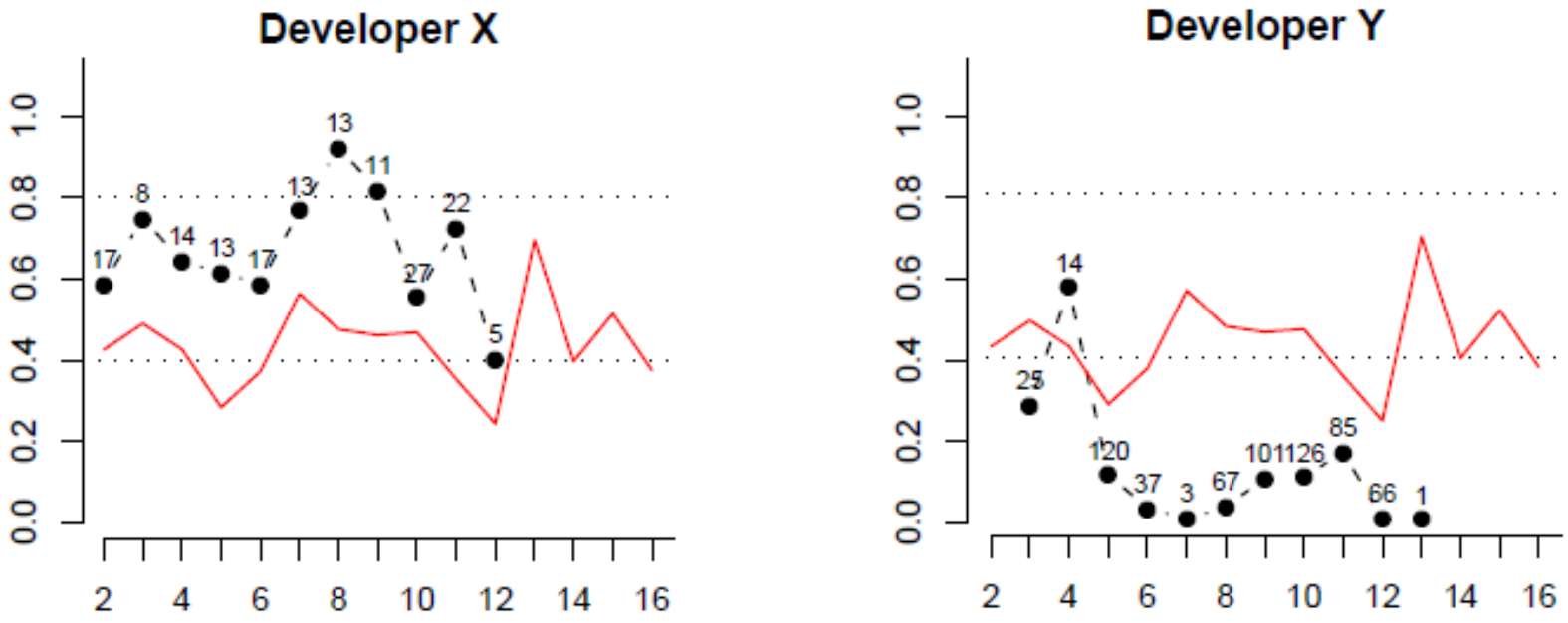
Provisioning 2 has 107 programmers.

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

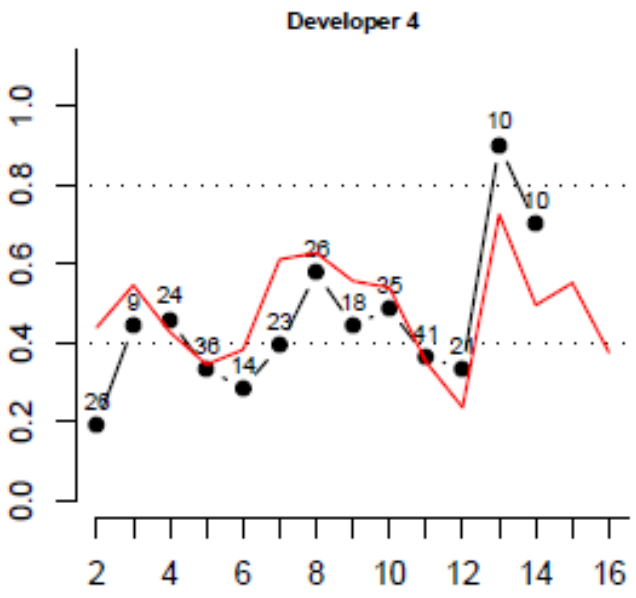
The average is about 0.4

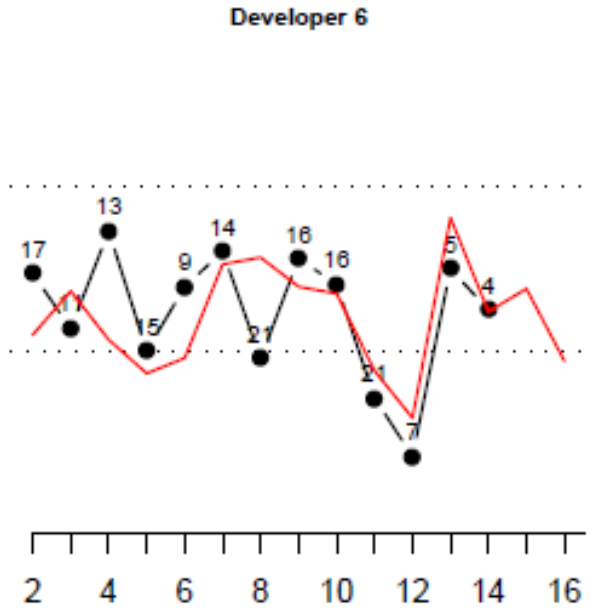


Average buggyfile ratio, all programmers



Buggyfile ratio

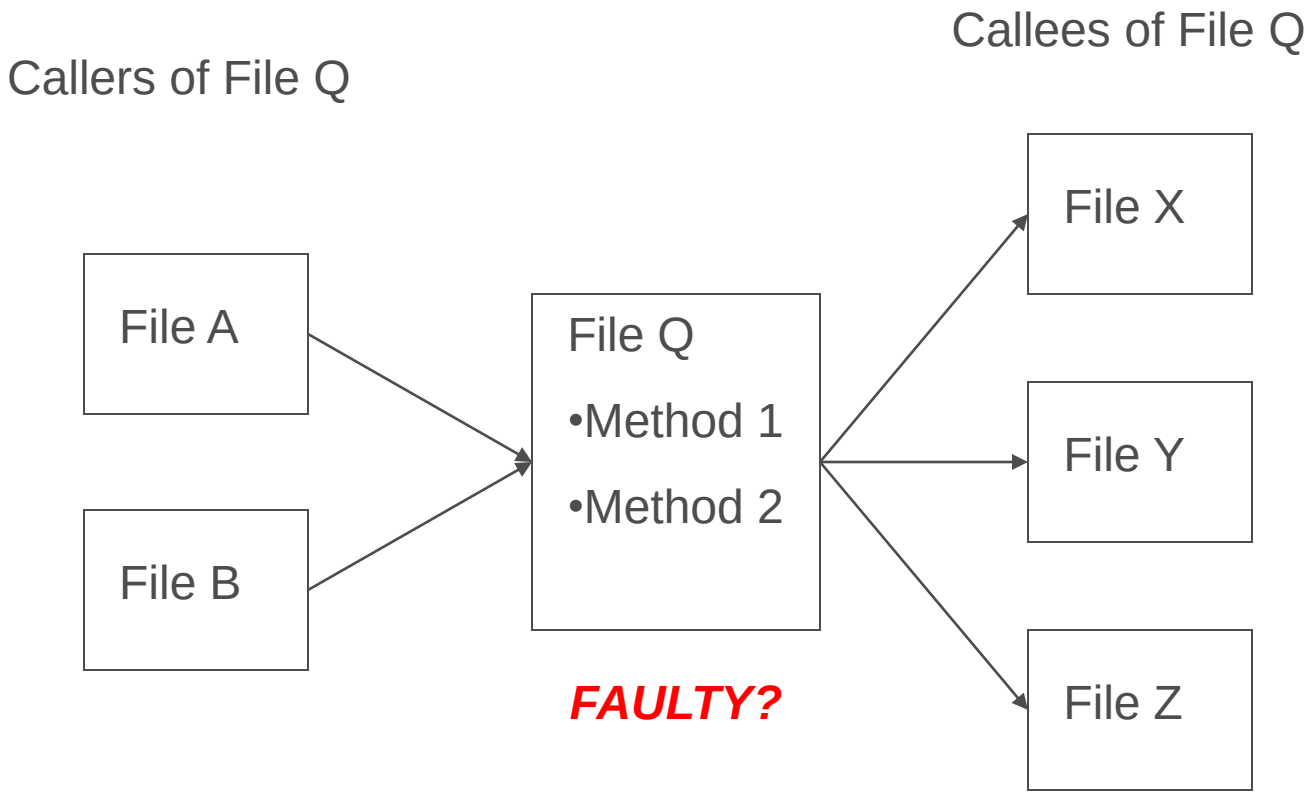




Buggyfile ratio
more typical cases

Calling Structure

- Are files that have high rate of interaction with other files more fault-prone?



Calling structure attributes that we investigated

For each file:

- number of callers & callees
- number of new callers & callees
- number of prior new callers & callees
- number of prior changed callers & callees
- number of prior faulty callers & callees
- ratio of internal calls to total calls

Fault prediction by multi-variable models

- Code and history attributes, no calling structure
- Code and history attributes, including calling structure
- Code attributes only, including calling structure

Method: Start with the single best attribute, and repeatedly add the attribute that most improves the prediction.

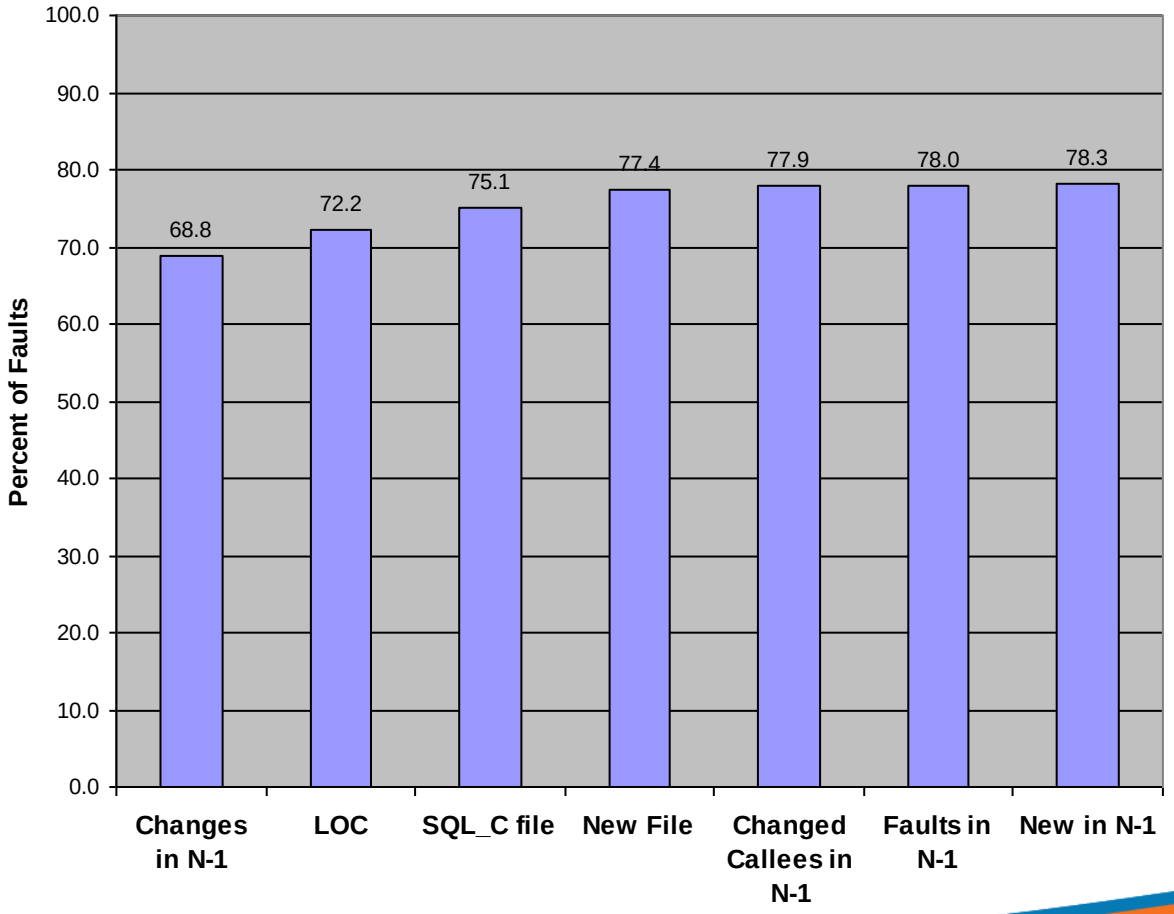
Forward Sequential Search

Model construction process

Initialize:
Model *M* consists solely of *RelNum*.
A = a set of candidate predictor attributes

***Add Variable*:**
for each attribute *a* in *A*,
    construct a model consisting of $M \cup \{a\}$.
    create predictions for the system.
    evaluate the likelihood ratio χ^2 statistic.
determine the single attribute a^* with the highest χ^2 statistic.
if the resulting P-value is >= 0.001, STOP. *M* is the final model.
else (i.e., P-value is < 0.001)
    add a^* to *M*.
    remove a^* from *A*.
    if *A* is empty, STOP. *M* is the final model.
    else repeat the *Add Variable* step.

Code and history attributes, including calling structure



Augmenting standard model with fine-grained change metrics

Change metrics in the standard model

- Changed/not changed
- Number of changes during a release

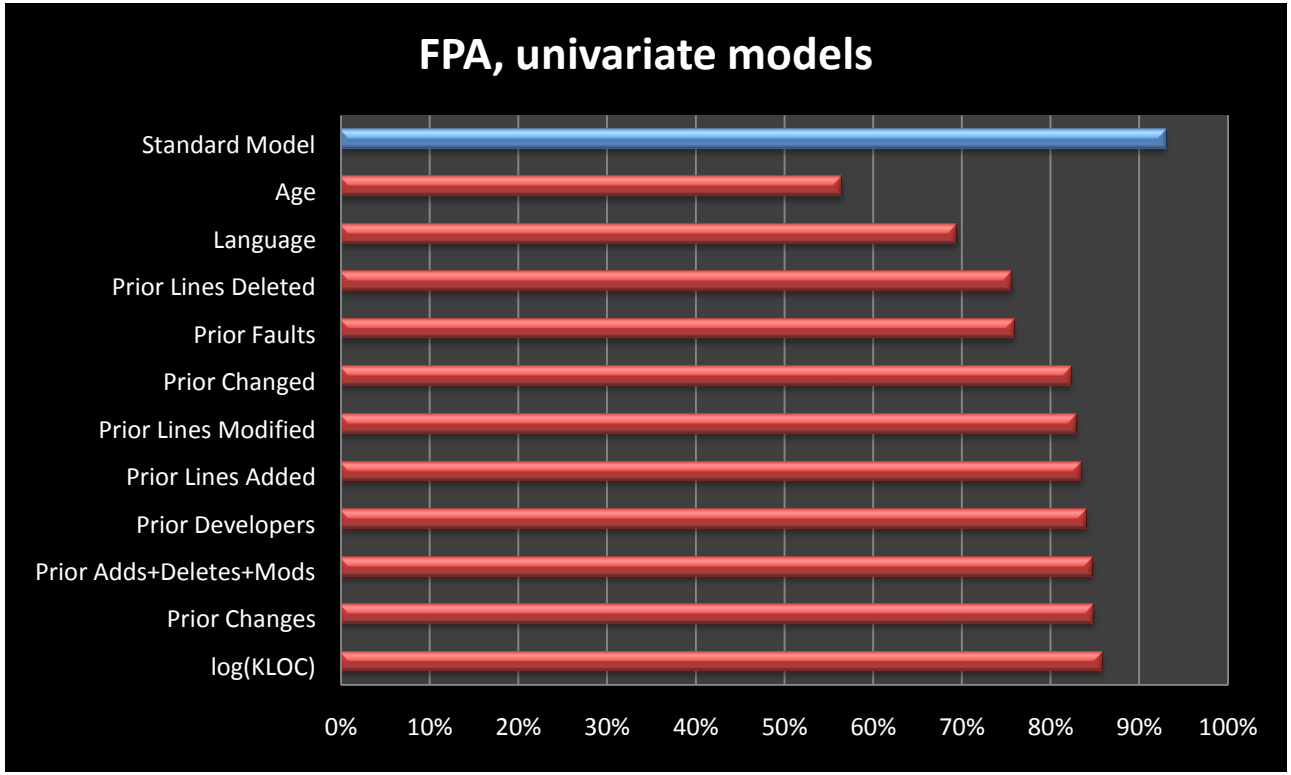
Do fine-grained change metrics provide more information than simple change counts?

- Number of lines added
- Number of lines deleted
- Number of lines modified
- Relative change (line changes/LOC)

One-step Forward Search

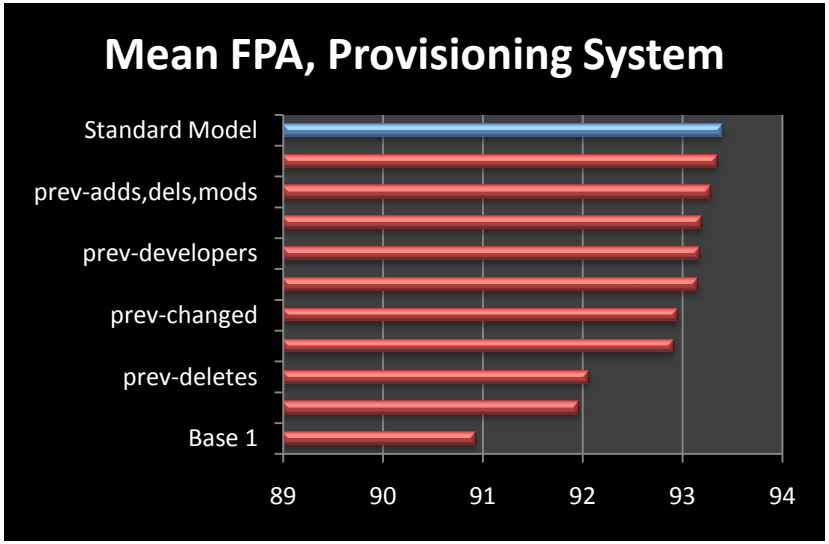
- We considered 8 different measures of change, both absolute and relative, and 3 versions (raw, square root, fourth root) of each as potential variables to augment a simplified version of the Standard Model.
- Each variable was first evaluated in a uni-variate model.
- Each of the 48 variables, and binary Changed/not Changed, was added separately to the Standard Model

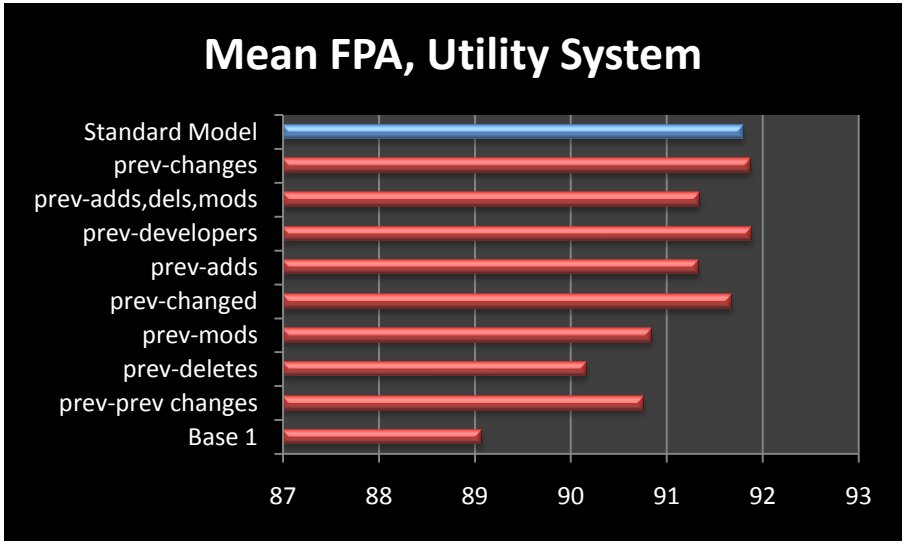
Fault-percentile averages for univariate predictor models: Provisioning 2 system (best result from raw variable, square root, fourth root)



Predictor Variables	Mean FPA	Standard error of increment (variation over releases)
Base (KLOC, Age, File type)	90.93	---
(Prior Changes)$^{1/2}$	93.35	0.24
(Prior Add+Deletes+Mods)$^{1/4}$	93.28	0.26
(Prior Add+Deletes+Mods/LOC)$^{1/4}$	93.19	0.28
(Prior Developers)$^{1/2}$	93.17	0.23
(Prior Lines Added)$^{1/4}$	93.15	0.26
(Prior Lines Added/LOC)$^{1/4}$	93.03	0.29
Prior Changed	92.95	0.23
(Prior Cum Developers)$^{1/2}$	92.93	0.17
(Prior Lines Modified)$^{1/4}$	92.91	0.20
(Prior Lines Modified/LOC)$^{1/4}$	92.81	0.20
(Prior Faults)$^{1/4}$	92.21	0.16
(Prior New Developers)$^{1/2}$	92.06	0.25
(Prior Lines Deleted)$^{1/4}$	92.06	0.16
(Prior Lines Deleted/LOC)$^{1/4}$	92.00	0.18
(Prior-Prior Changes)$^{1/4}$	91.96	0.14

Base Model, and added variables





- Base model
  - KLOC
  - File age (number of releases)
  - File type (C,C++,java,sql,make,sh,perl,...)

Is SBSE in our future?

- Finding optimal set of attributes for prediction model
- Looking for the best way to make use of individual developer information

That's all folks

Questions?