Mining Metrics to Predict Component Failures

Nachiappan Nagappan, Microsoft Research
Thomas Ball, Microsoft Research
Andreas Zeller, Saarland University

Overview

- Introduction
- Hypothesis and high level description
- Background work and contributions
- Case study results
- Conclusions and future work

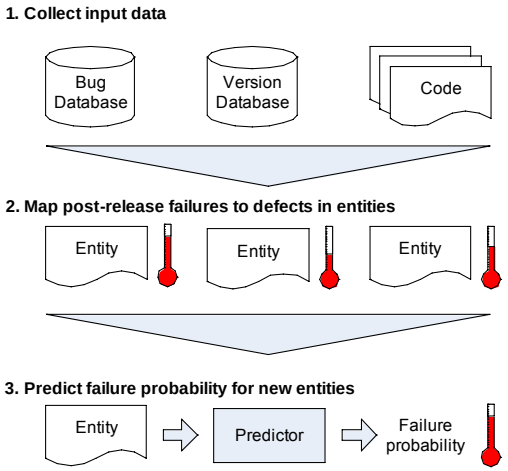
Introduction

- Software quality assurance in large organizations requires considerable effort
- Helping focus such activities based on early estimates of software quality is very useful
- Such estimations can be based on a variety of factors,
  - Previous failures
  - Evolution metrics
  - Static code complexity metrics
- In this talk we focus on static code complexity measures

Hypothesis

	Hypothesis
H_1	Increase in complexity metrics of an entity *E* correlates with the number of post-release defects of *E*.
H_2	There is a common subset of metrics for which H_1 applies in all projects.
H_3	There is a combination of metrics which significantly predicts the post-release defects of new entities within a project.
H_4	Predictors obtained using H_3 from one project also predict failure-prone entities in other projects.

High level description



Background work

- Defects and failures
  - *defect* refers to an error in the source code, and *failure* refers to an observable error in the program behavior
  - every failure can be traced back to some defect, but a defect need not result in a failure
  - For the user, only post-release failures matter. Our approach is exclusively concerned with *post-release defects*, each of them uncovered by at least one failure in the field.
- Complexity metrics
  - Basili et al. Eight student projects – LCOM not correlated with defects.

Background work

- Subramanyam and Krishnan survey eight empirical studies, all showing that OO metrics are significantly associated with defects.
- Historical Data
  - Hudepohl et al. successfully predicted whether a module would be defect-prone or not by combining metrics and historical data. They used *software design metrics* as well as *reuse information*, under the assumption that new or changed modules would have a higher defect density.

Contributions

- It reports on how to systematically build predictors for post-release defects from failure history found in the field by customers.
- It investigates whether object-oriented metrics can predict post-release defects from the field.
- It analyzes whether predictors obtained from one project history are applicable to other projects.
- It is one of the largest studies of commercial software—in terms of code size, team sizes, and software users.

Projects researched

- Internet Explorer 6
- IIS Server
- Windows Process Messaging
- DirectX
- NetMeeting

>1,000,000 Lines of Code

Projects researched

A B C D E

We randomize the projects as A, B, C, D, E

Research questions

- Do metrics correlate with failures?
- Is there a set of metrics that fits all projects?
- Can we predict “failure-prone” modules?
- Are predictors applicable across projects?

Metrics and their Correlation with Post-Release Defects

Metric	Description		Correlation with post-release defects of *M*				
			A	B	C	D	E
Module metrics — correlation with metric in a module *M*							
Classes	# Classes in *M*		**0.531**	**0.612**	**0.713**	0.066	0.438
Function	# Functions in *M*		0.131	**0.699**	**0.761**	0.104	**0.531**
GlobalVariables	# global variables in *M*		0.023	**0.664**	**0.695**	0.108	0.460

Spearman rank correlation
(bold: significant at 0.05)

Per-function metrics — correlation with maximum and sum of metric across all functions *f()* in a module *M*							
Lines	# executable lines in *f()*	Max	-0.236	**0.514**	**0.585**	**0.496**	**0.509**
		Total	0.131	**0.709**	**0.797**	0.187	**0.506**
Parameters	# parameters in *f()*	Max	-0.344	0.372	**0.547**	0.015	0.346
		Total	0.116	**0.689**	**0.790**	0.152	0.478
Arcs	# arcs in *f()*'s control flow graph	Max	-0.209	0.376	**0.587**	0.527	0.444
		Total	0.127	**0.679**	**0.803**	0.158	**0.484**
Blocks	# basic blocks in *f()*'s control flow graph	Max	-0.245	0.347	**0.585**	0.546	0.462
		Total	0.128	**0.707**	**0.787**	0.158	0.472
ReadCoupling	# global variables read in *f()*	Max	-0.005	**0.582**	**0.633**	0.362	0.229
		Total	-0.172	**0.676**	**0.756**	0.277	0.445
WriteCoupling	# global variables written in *f()*	Max	0.043	**0.618**	**0.392**	0.011	0.450
		Total	-0.128	**0.629**	**0.629**	0.230	0.406
AddrTakenCoupling	# global variables whose address is taken in *f()*	Max	0.237	**0.491**	**0.412**	0.016	0.263
		Total	0.182	**0.593**	**0.667**	0.175	0.145
ProcCoupling	# functions that access a global variable written in *f()*	Max	-0.063	**0.614**	**0.496**	0.024	0.357
		Total	0.043	**0.562**	**0.579**	0.000	0.443
FanIn	# functions calling *f()*	Max	0.034	**0.578**	**0.846**	0.037	**0.530**
		Total	0.066	**0.676**	**0.814**	0.074	**0.537**
FanOut	# functions called by *f()*	Max	-0.197	0.360	**0.613**	0.345	0.465
		Total	0.056	**0.651**	**0.776**	0.046	**0.506**
Complexity	McCabe's cyclomatic complexity of *f()*	Max	-0.200	0.363	**0.594**	0.451	**0.543**
		Total	0.112	**0.680**	**0.801**	0.165	**0.529**

Metrics and their Correlation with Post-Release Defects

Per-class metrics — correlation with maximum and sum of metric across all classes *C* in a module *M*							
ClassMethods	# methods in *C* (private / public / protected)	Max	0.244	**0.589**	**0.534**	0.100	0.283
		Total	**0.520**	**0.630**	**0.581**	0.094	0.469
InheritanceDepth	# of superclasses of *C*	Max	**0.428**	**0.546**	0.303	0.131	0.323
		Total	**0.432**	**0.606**	**0.496**	0.111	0.425
ClassCoupling	# of classes coupled with *C* (e.g. as attribute / parameter / return types)	Max	**0.501**	**0.634**	**0.466**	-0.303	0.264
		Total	**0.547**	**0.598**	**0.592**	-0.158	0.383
SubClasses	# of direct subclasses of *C*	Max	0.196	**0.502**	**0.582**	-0.207	0.387
		Total	0.265	**0.560**	**0.566**	-0.170	0.387

Do metrics correlate with failures?

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	almost all
C	all except *MaxInheritanceDepth*
D	only *#Lines (software was refactored if metrics indicated a problem)*
E	*#Functions, #Arcs, Complexity*

Do metrics correlate with failures?

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	al st a
C	excep axInheritance
D	only #Lin
E	#Functio mpl

YES

Is there a set of metrics that fits all projects?

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	almost all
C	all except *MaxInheritanceDepth*
D	only *#Lines*
E	*#Functions, #Arcs, Complexity*

Is there a set of metrics that fits all projects?

Project	Metrics correlated w/ failure
A	*#Classes* and 5 derived
B	ost al
C	ept axI ritanceDe
D	nly #
E	*#Functions, #Arcs, Complexity*

NO

Can we predict failure-prone modules?

- Basic idea: *Combine metrics*
- Problem: Metrics are *intercorrelated*
- Solution: Principal Component Analysis (PCA)

Principal Components

Project E

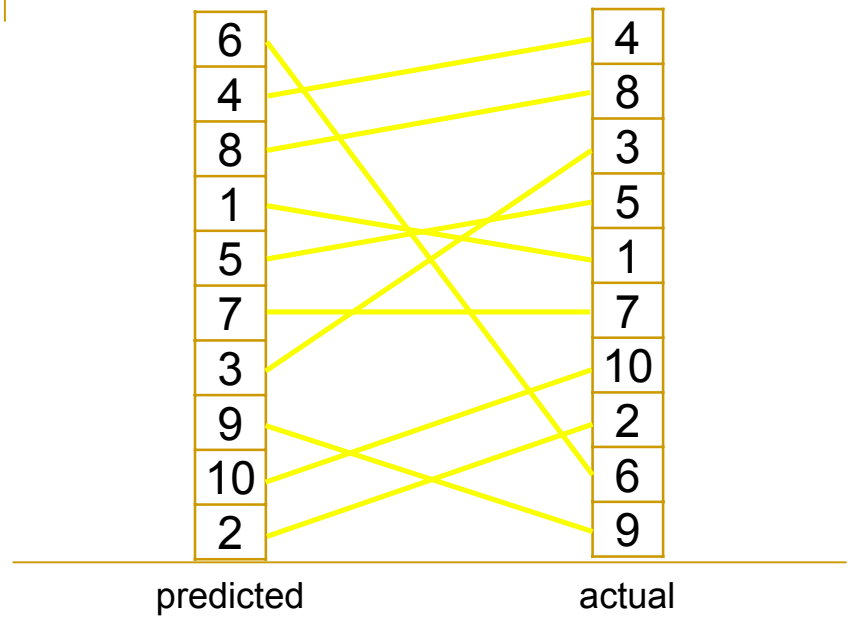
Component	%Variance	Cumul %
1	76.6	76.6
2	9.2	85.8
3	6.2	92.0
4	2.8	94.7
5	1.6	96.3

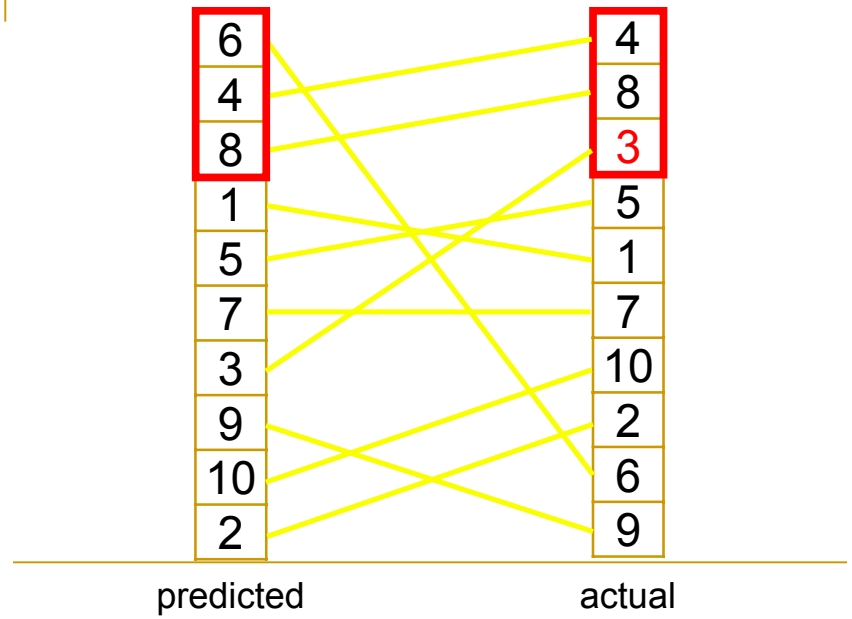
Predictive Power

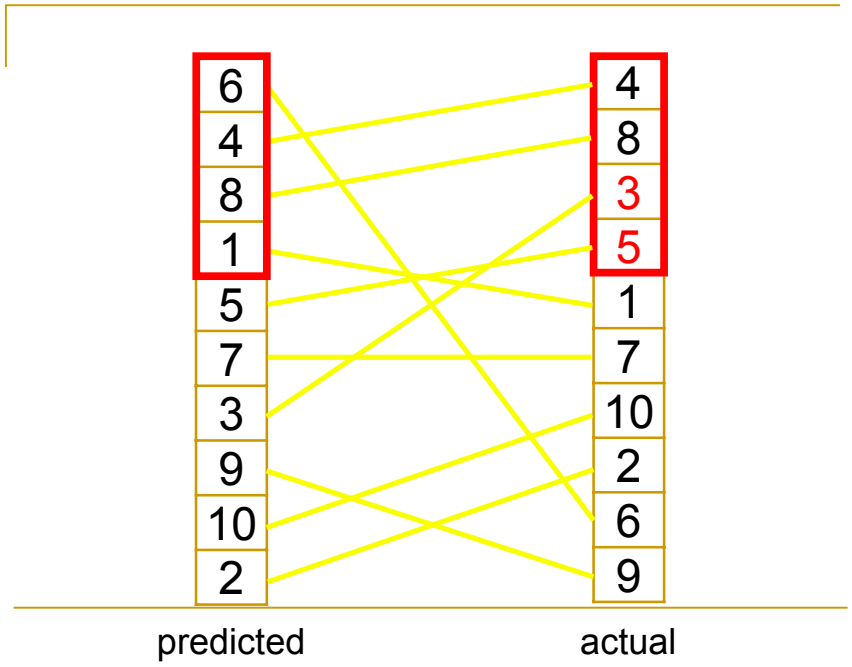
- From the principal components, we can build *regression models*
- These can be fed with metrics to predict the defect likelihood
- Modules can be *ranked* according to likelihood

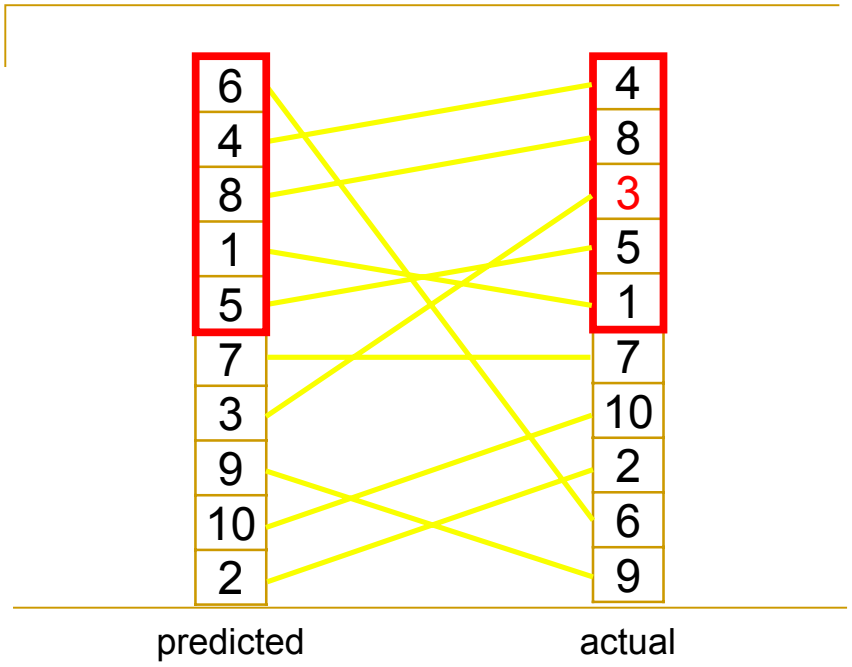
A Ranking

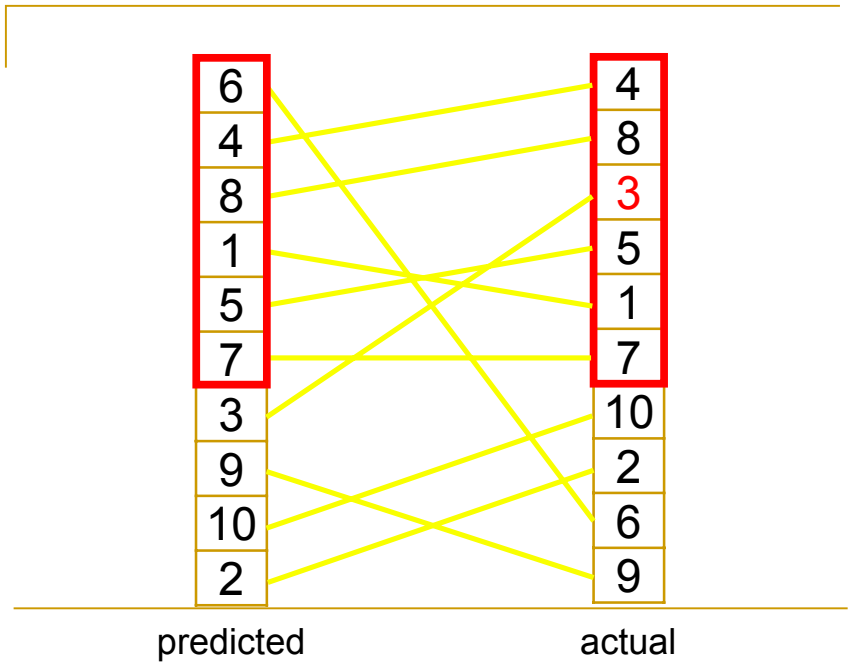
- 1/3 of the modules
- ranked according to predictor built from 2/3 of the modules
- can be *evaluated* against actual ranking

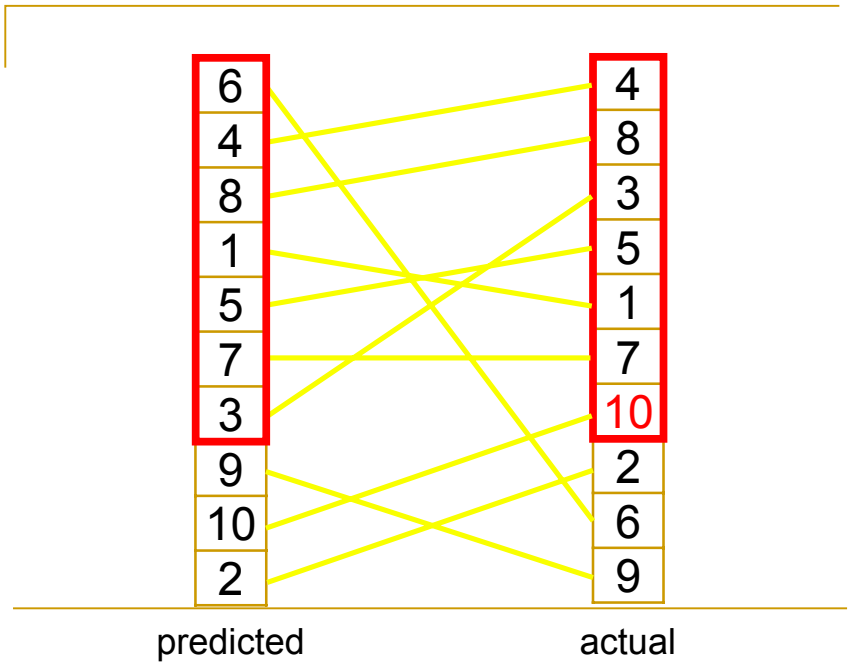





6
4
8
1
5
7
3
9
10
2
4
8
3
5
1
7
10
2
6
9
predicted
actual


6
4
8
1
5
7
3
9
10
2
4
8
3
5
1
7
10
2
6
9
predicted
actual


6
4
8
1
5
7
3
9
10
2
4
8
3
5
1
7
10
2
6
9
predicted
actual


6
4
8
1
5
7
3
9
10
2
4
8
3
5
1
7
10
2
6
9
predicted
actual

Predictive Power

Project	#Components	R^2 value
A	9	0.741
B	6	0.779
C	7	0.579
D	7	0.684
E	5	0.919

Can we predict failure-prone modules?

Project	#Components	R^2 value
A	9	0.741
B		
C		
D		
E	5	0.919

YES

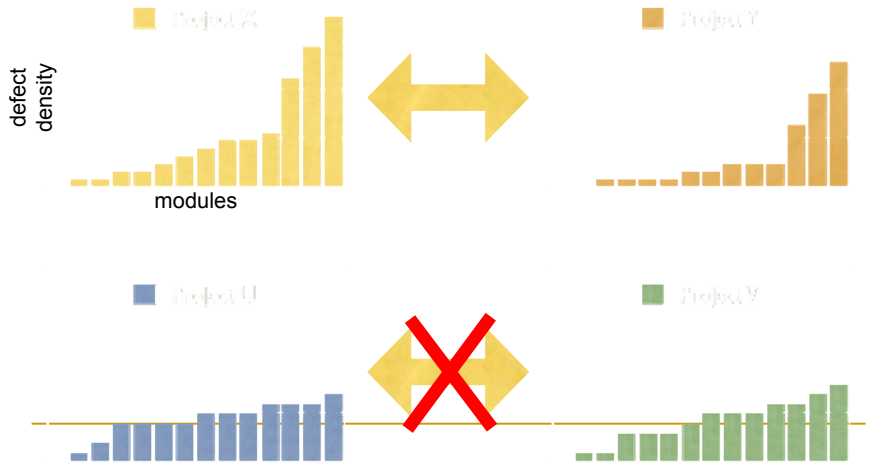
Are predictors applicable across projects?

Project	A	B	C	D	E
A	•	no	no	no	no
B	no	•	**yes**	no	no
C	no	**yes**	•	no	(yes)
D	no	no	no	•	no
E	no	no	(yes)	no	•

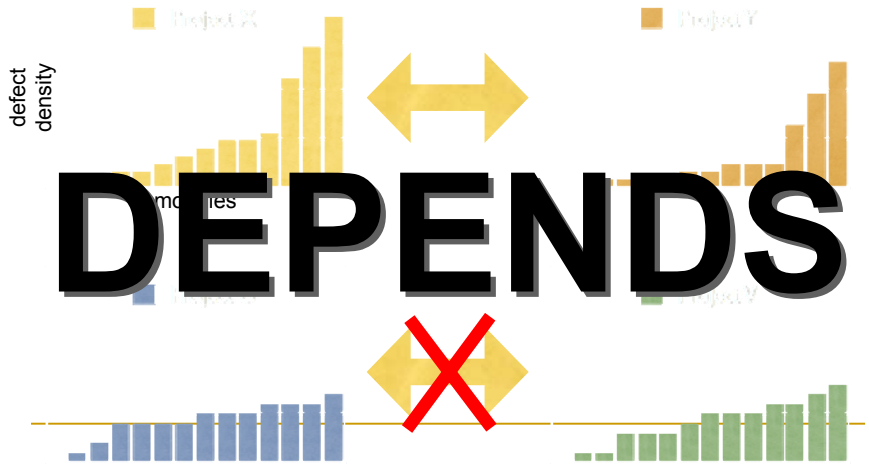
Predictive Projects

- Predictors learned from project B are applicable to project C and vice versa
- Project B and C both have a *heterogeneous defect distribution:*
  - Few modules have all the failures…
  - and those modules are complex, too
- This leads to a good predictive power

Are predictors applicable across projects?



Are predictors applicable across projects?



Summary

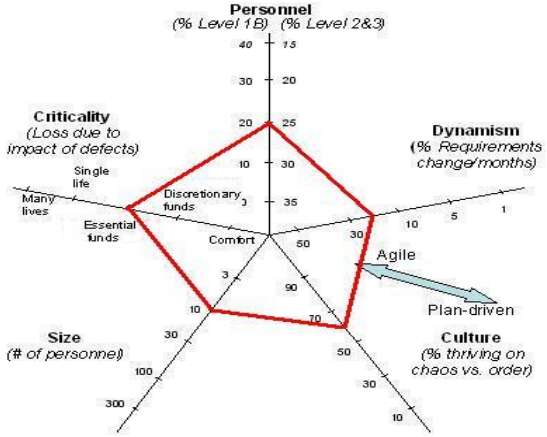
- Metrics correlate with failures
- There is no set of metrics that fits all projects
- We can predict failure-prone modules
- Predictors are applicable across similar projects

Discussed results with teams to drive future work

Future work

More metrics	More projects
More automation	More data

Process similarity



Appendix: Cyclomatic Complexity

- Given a program's control flow graph, McCabe's Cyclomatic Complexity M is defined as $M = E - N + 2P$
  - where
    - E = the number of edges of the graph
    - N = the number of nodes of the graph
    - P = the number of connected components.
- M is equivalently defined to be one larger than the number of decision points (if/case-statements, while-statements, etc).
- Separate functions or methods are treated as being independent, disconnected components of the program's control flow graph.