MSR Approaches


Defect prediction
Change analysis and prediction
Empirical studies
Expertise & bug triaging
Visual evolution analysis
Human factors

Defect prediction

Change analysis and prediction

Empirical studies

Expertise & bug triaging

Visual evolution analysis

Human factors

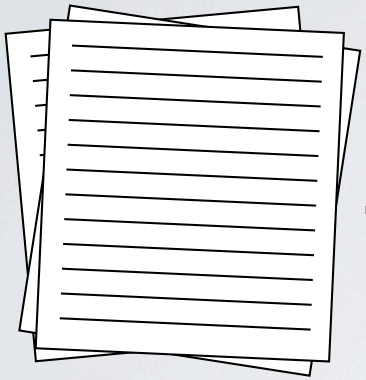
Defect prediction

- Goal: Optimize available QA resources by
  - reducing code review cost
  - identifying defect-prone components
  - ranking component according to defect proneness

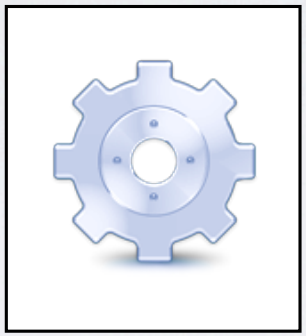
Defect prediction in a nutshell

Coarse grained prediction: module/ binary/subsystem

Fine grained prediction: class/file



Prediction data



Prediction model

Classification

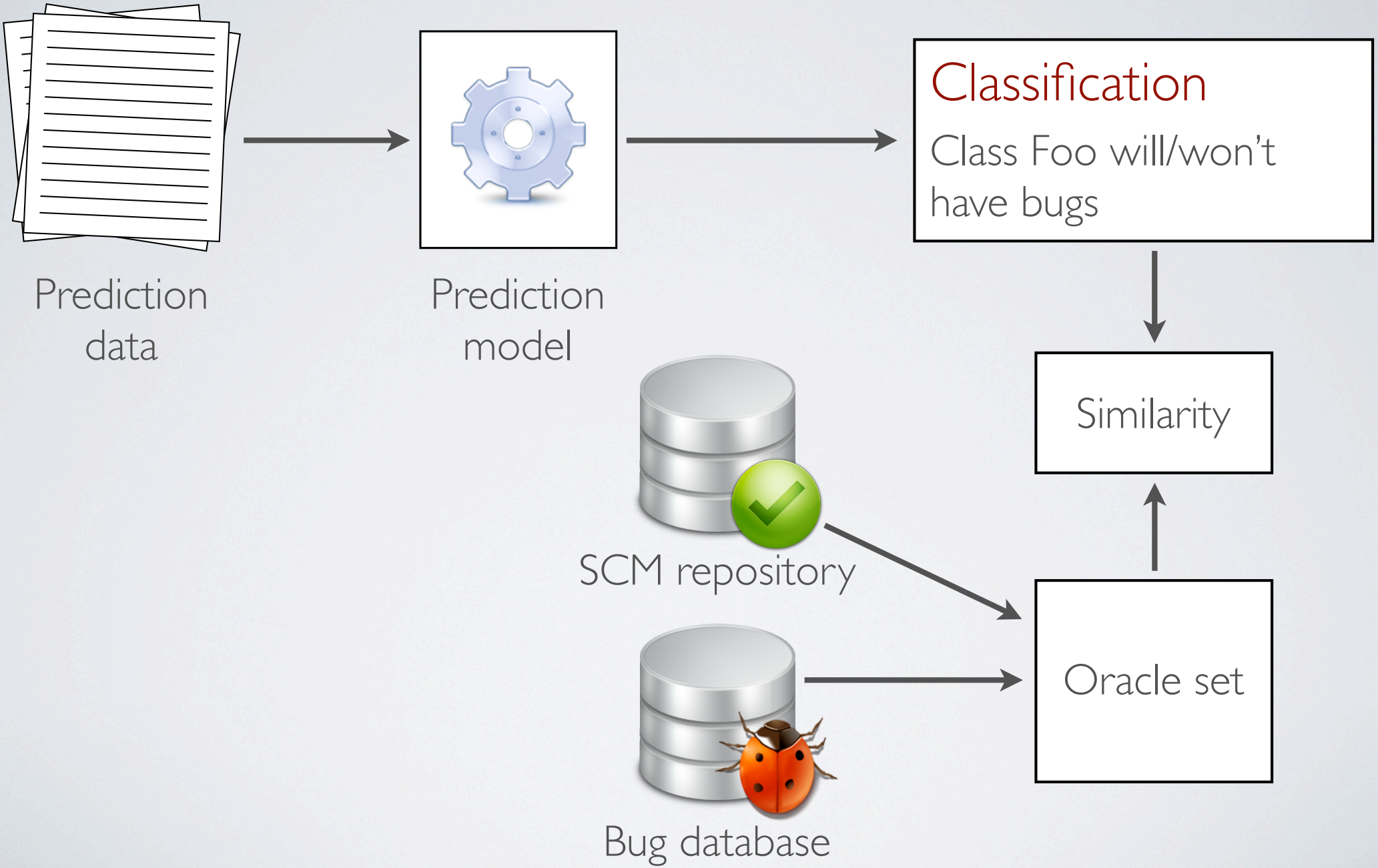
Class Foo will/won't have bugs

Correct?

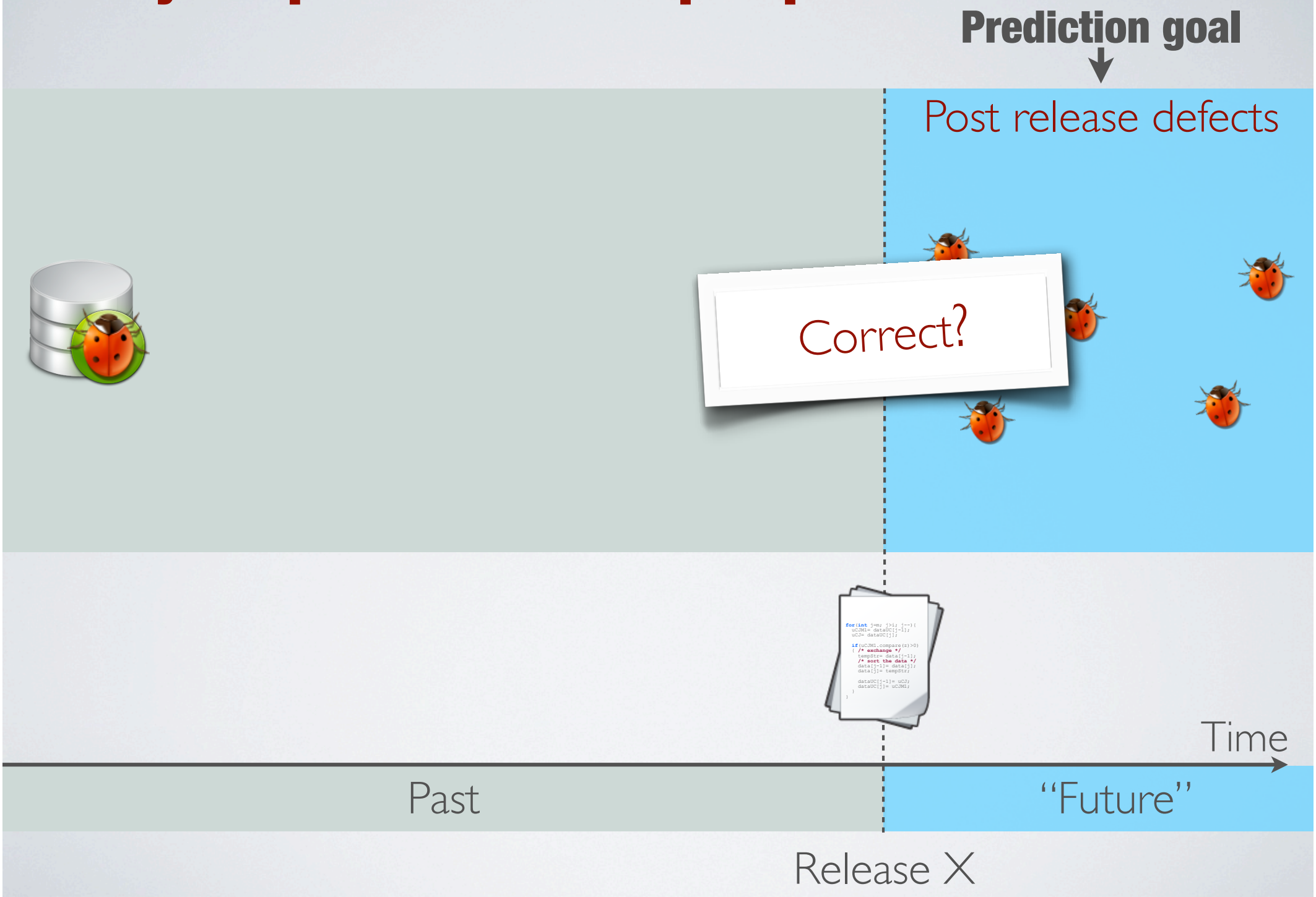
Ranking

Class Foo will have more bugs than class Bar

How to verify the prediction



Verify the prediction: time perspective

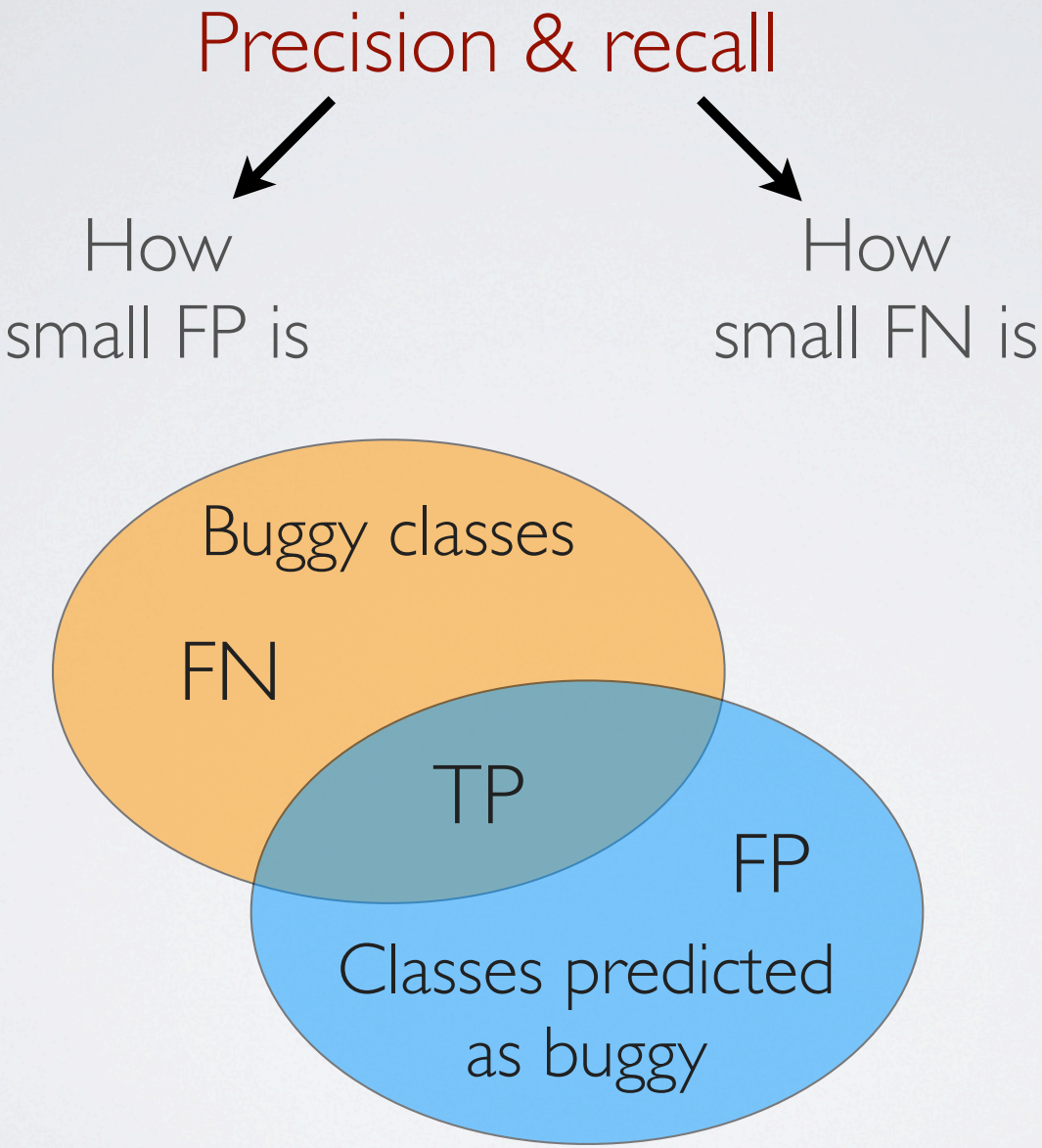


Evaluating defect prediction approaches

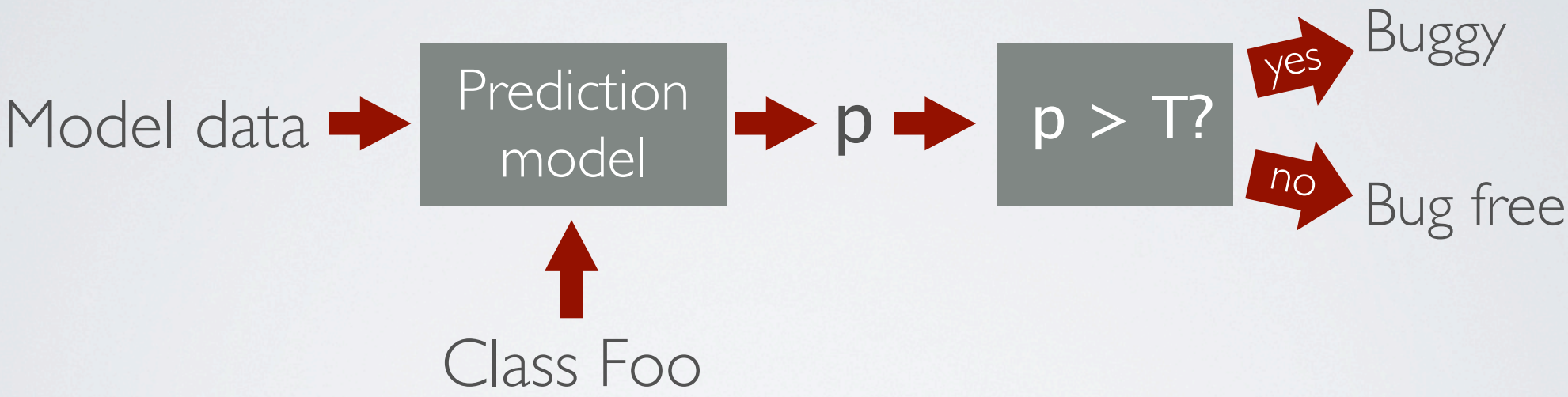
$$\text{Accuracy} = \frac{\text{Size(Correctly classified artifacts)}}{\text{Size(All artifacts)}}$$

- Defects distribution typically follows the Pareto law
  - 80% of defects are found in 20% of artifacts
  - A trivial model that predicts all artifact as defect prone would provide a very good accuracy

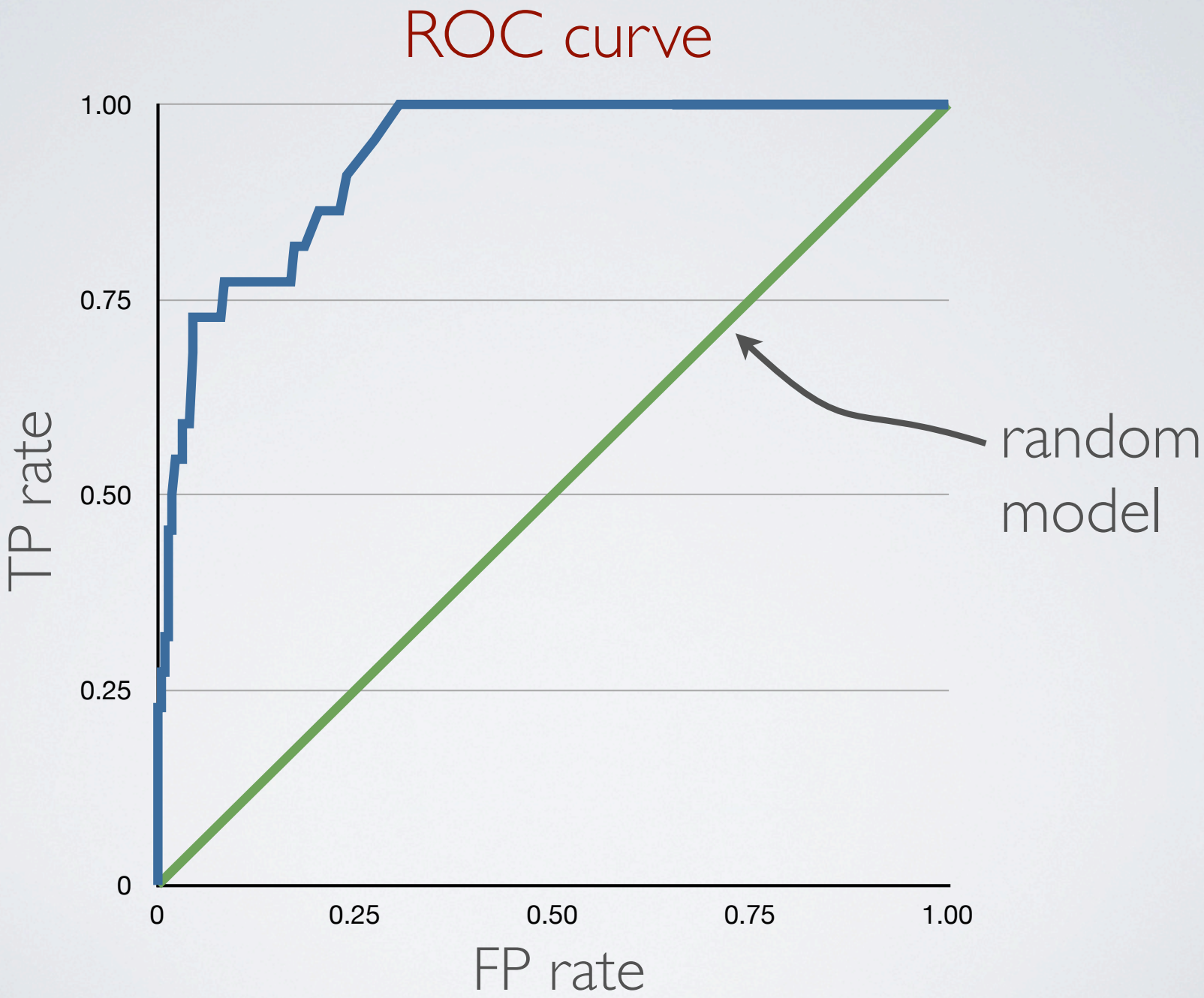
Evaluating classification



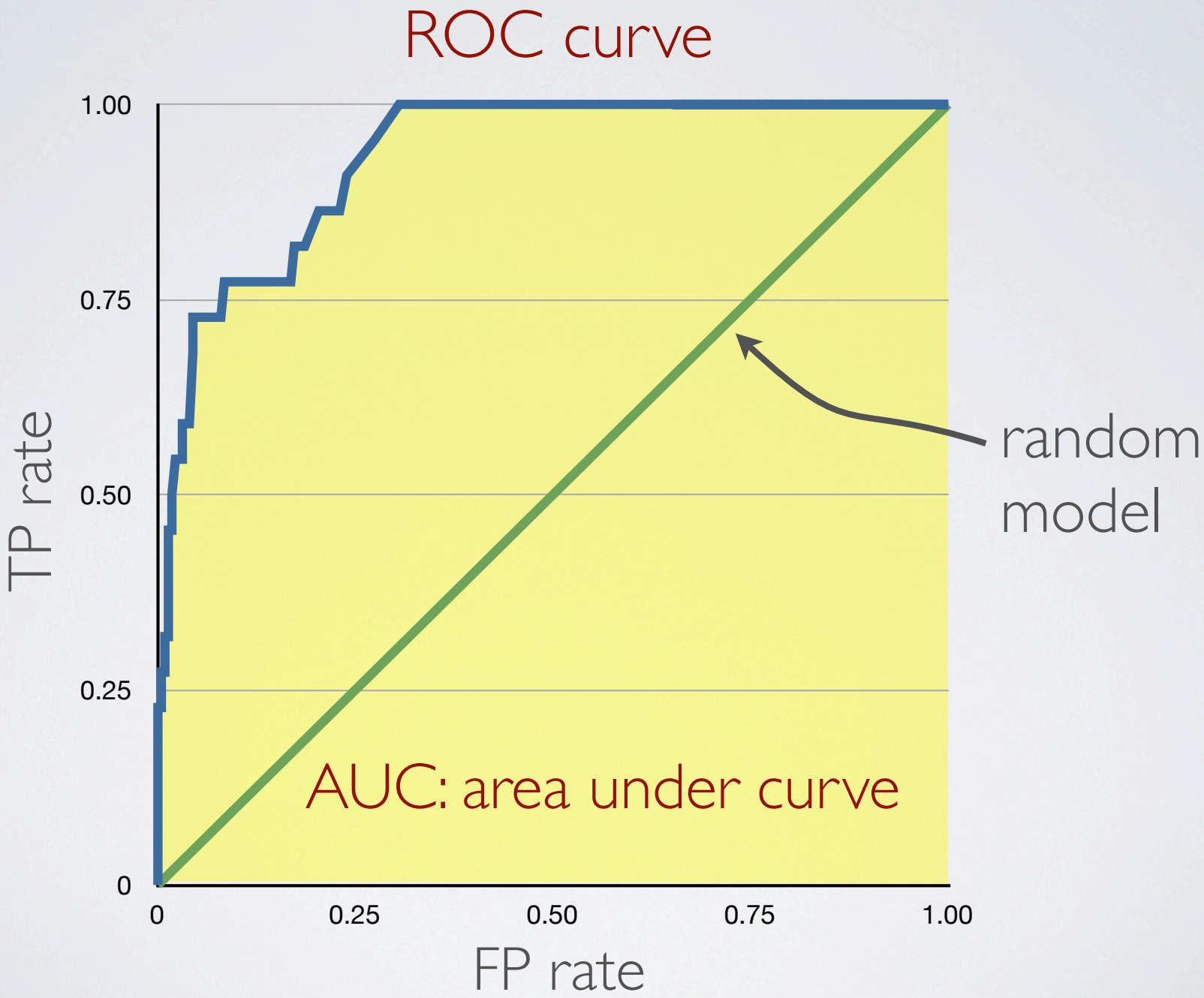
Evaluating classification



Evaluating classification

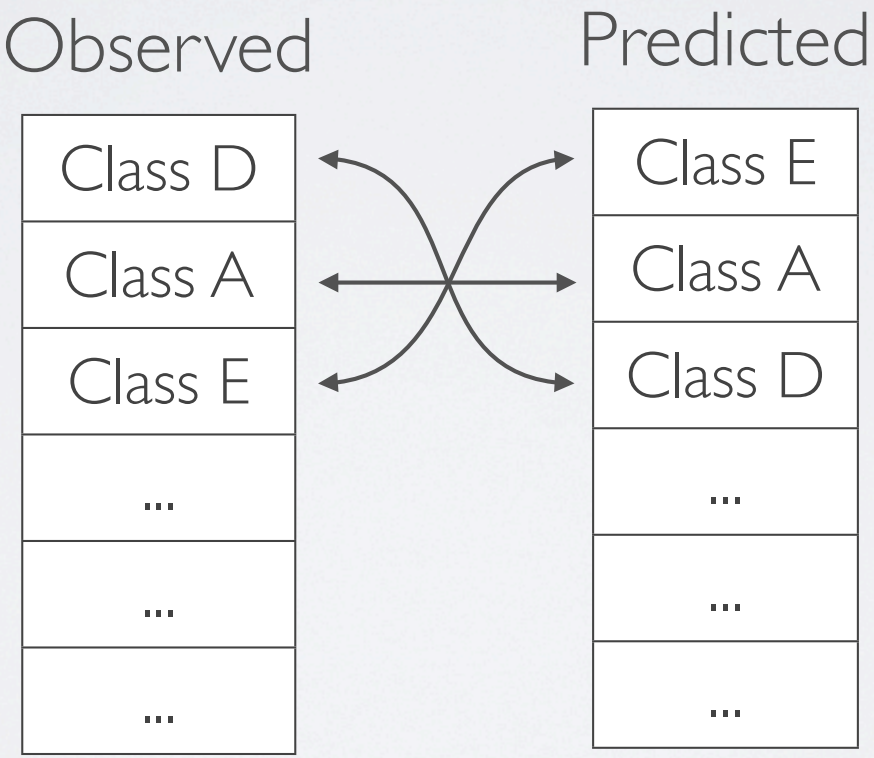


Evaluating classification



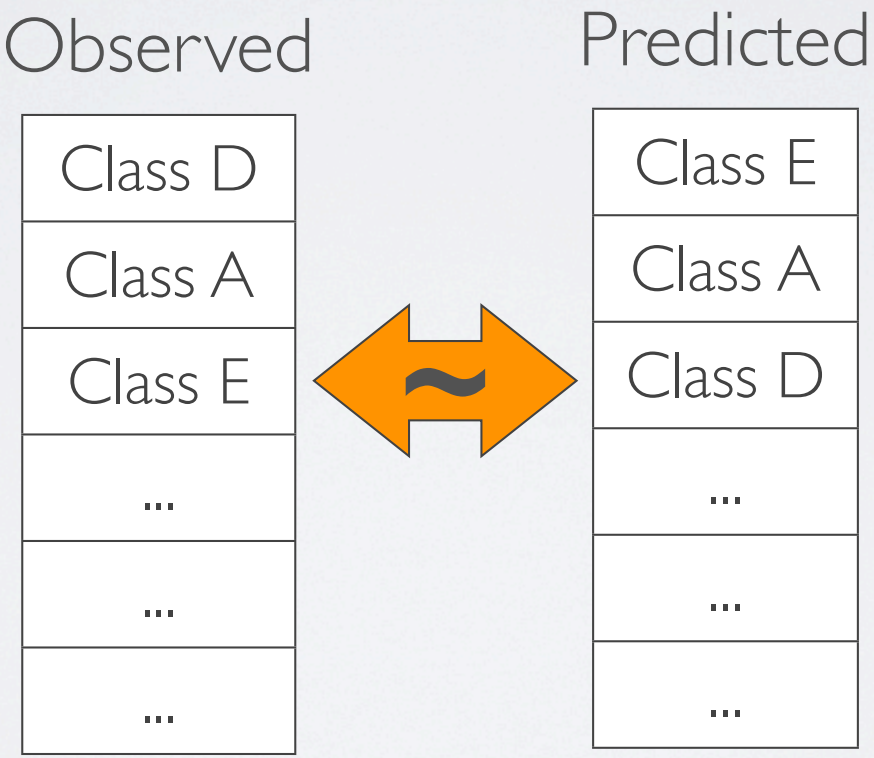
Evaluating ranking

Spearman's correlation coefficient

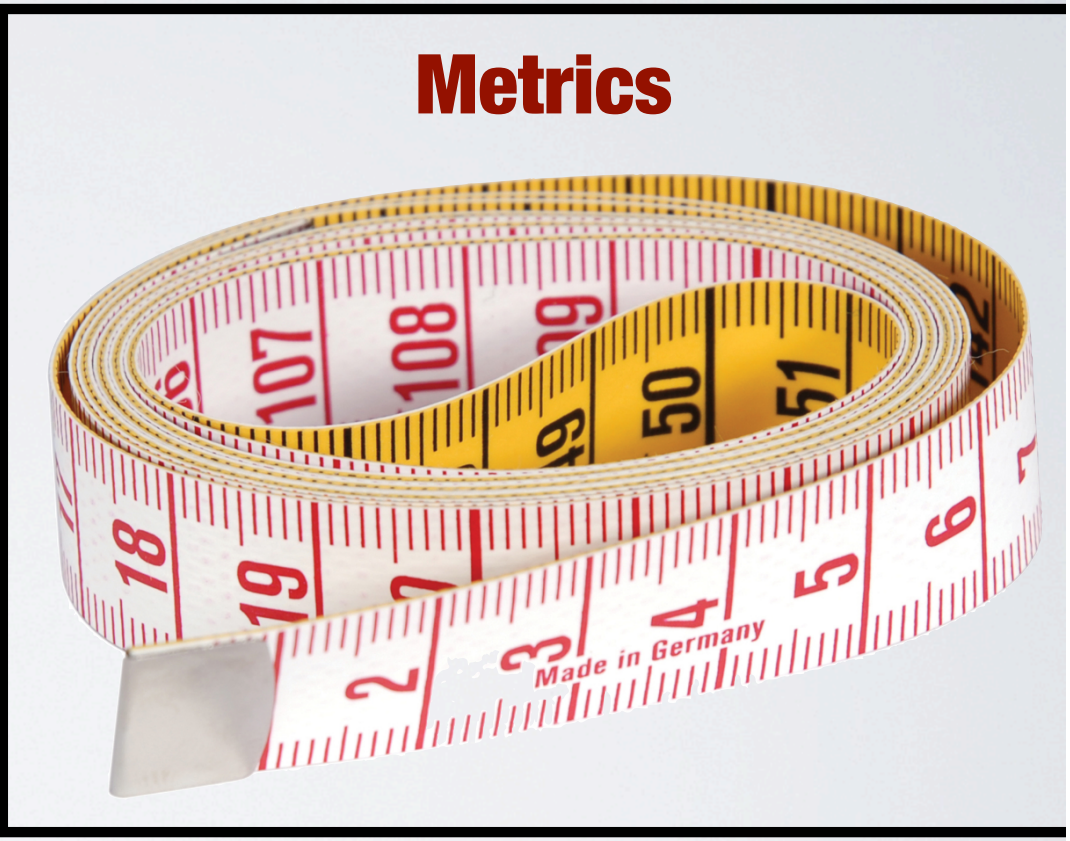


Evaluating ranking

Spearman's correlation coefficient

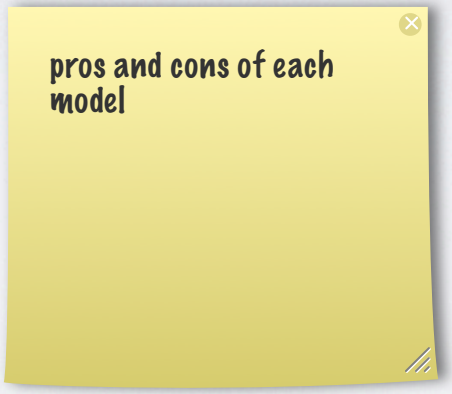


Approaches are based on:



Typical prediction models

- Linear or logistic regression
- Decision trees - Random Forest
- Naïve Bayes
- SVM



Predicting defects with LOC

- A number of studies demonstrated that LOC is highly correlated with defects and can be used as a predictor for future defects
- The hypothesis is that larger (or more complex) artifacts are more likely to introduce defects

Basili et al., *A validation of object-oriented design metrics as quality indicators*, TSE 1996

Graves et al., *Predicting Fault Incidence Using Software Change History*, TSE 2000

Gyimóthy et al., *Empirical validation of object-oriented metrics on open source software for fault prediction*, TSE 2005

Ostrand et al., *Predicting the Location and Number of Faults in Large Software Systems*, TSE 2005

Predicting defects with changes

- Code churn and number of changes were largely employed as predictors for defects
- The hypothesis is that frequently modified artifacts are more likely to introduce defects

Khoshgoftaar et al., Detection of software modules with high debug code churn in a very large legacy system, ISSRE 1996

Graves et al., *Predicting fault incidence using software change history*, TSE 2000

Nagappan and Ball, *Use of Relative Code Churn Measures to Predict System Defect Density*, ICSE 2005

Moser et al., *A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction*, ICSE 2008

Hassan , *Predicting faults using the complexity of code changes*, ICSE 2009

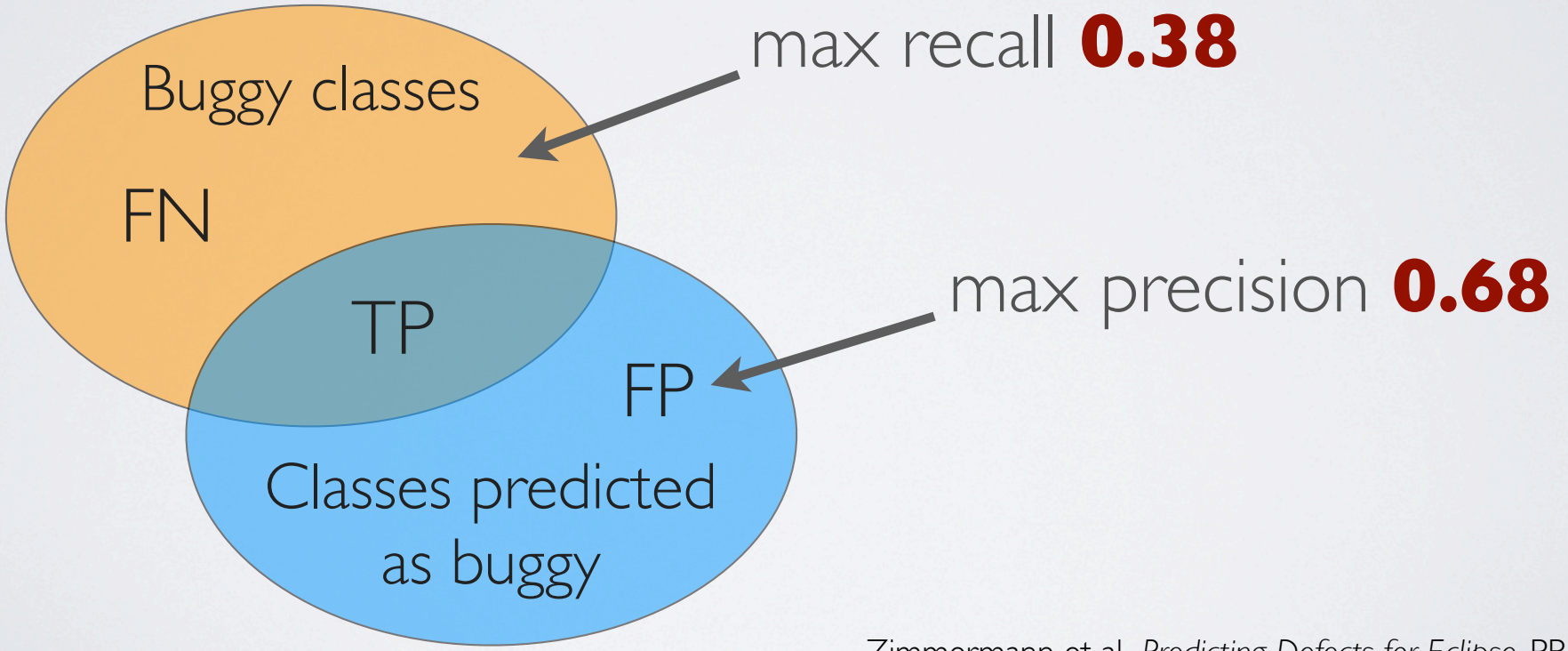
Predicting Defects for Eclipse

- Classification and ranking on 3 versions of Eclipse

Zimmermann et al., *Predicting Defects for Eclipse*, PROMISE 2007

Predicting Defects for Eclipse

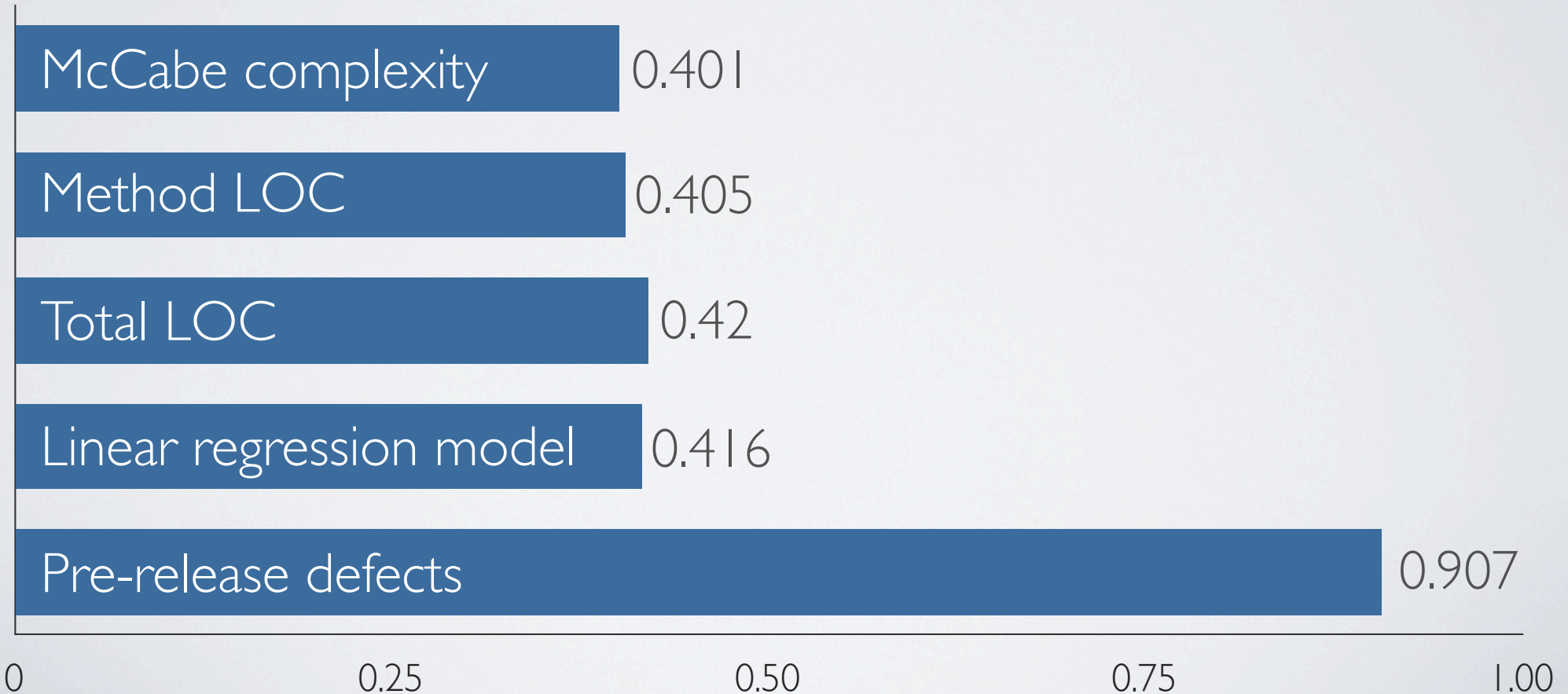
- Classification and ranking on 3 versions of Eclipse
- Classification with logistic regression



Zimmermann et al., *Predicting Defects for Eclipse*, PROMISE 2007

Predicting Defects for Eclipse

- Classification and ranking on 3 versions of Eclipse
- Ranking (Spearman's correlation)



Predicting Defects for Eclipse

- Classification and ranking on 3 versions of Eclipse
- Ranking (Spearman's correlation)

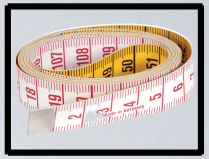
Temporal locality
previous defects is a good predictor for future defects*

McCabe cor
Method LOC
Total LOC 0.42
Linear regression model 0.416
Pre-release defects 0.907

0 0.25 0.50 0.75 1.00

*Ostrand et al., *The distribution of faults in a large industrial software system*, ISSTA 2002

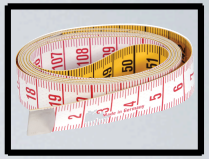
Mining metrics to predict component failures



- Classification and ranking on 5 Microsoft products
- Complexity metrics do correlate with defects
- There is no unique set of metrics that predicts defects in all projets
- Metrics can be combined (linear/logistic regression) to predict defect but
  - A predictor obtained from one project is not applicable to another project

Nagappan et al., *Mining metrics to predict component failures,* ICSE 2006

Mining metrics to predict component failures



- Classification and ranking on 5 Microsoft products
- Complexity metrics do correlate with defects
- There is no single set of metrics that fits all projects
- Metrics can be combined (linear/logistic regression) to predict defect but
  - A predictor obtained from one project is not applicable to another project

metrics can be used to predict defects, but they must be validated on the history

Nagappan et al., *Mining metrics to predict component failures*, ICSE 2006

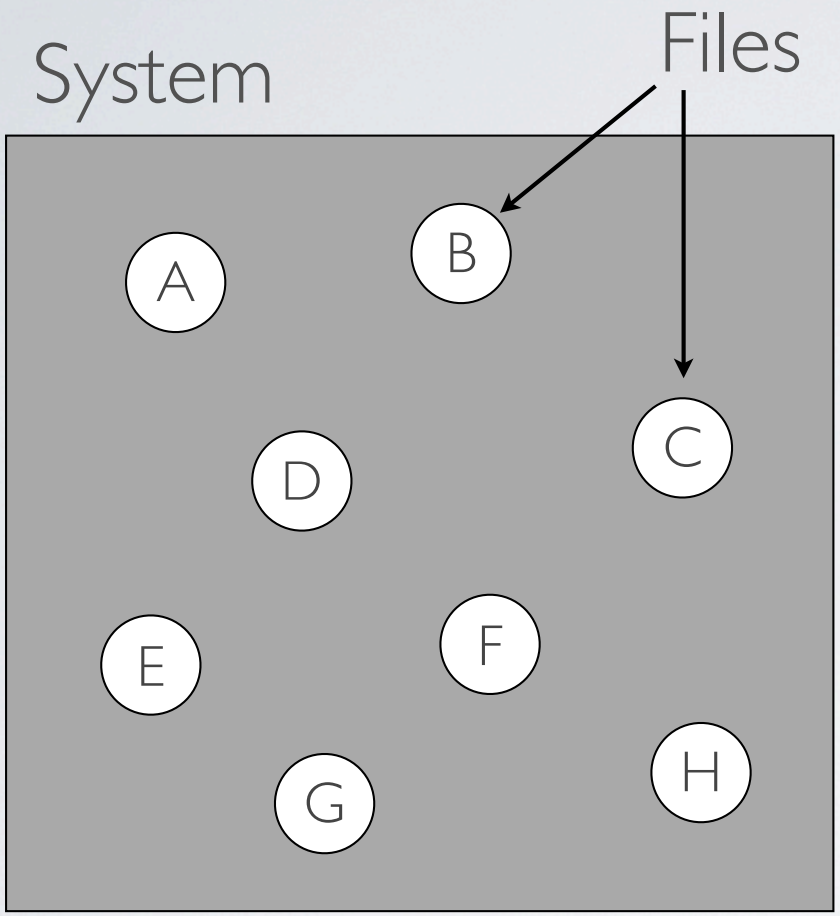
Predicting defects with BugCache

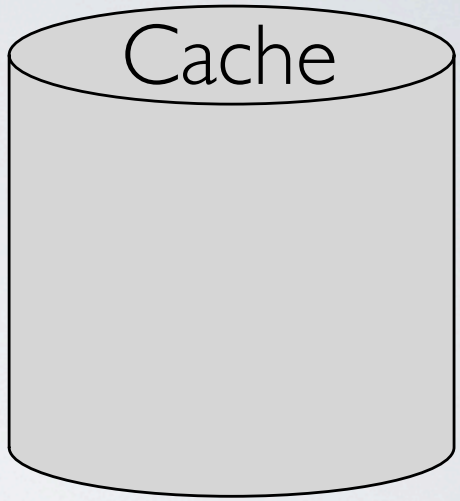
- Online prediction approach based on 5 principles:

1. Spatial locality: bug-prone artifacts are changed together
2. Temporal locality: artifacts which exhibit bugs in the past are likely to exhibit them also in the future
3. Bugs are found in new artifacts
4. Bugs are found in frequently modified artifacts
5. Bugs are found in large artifacts

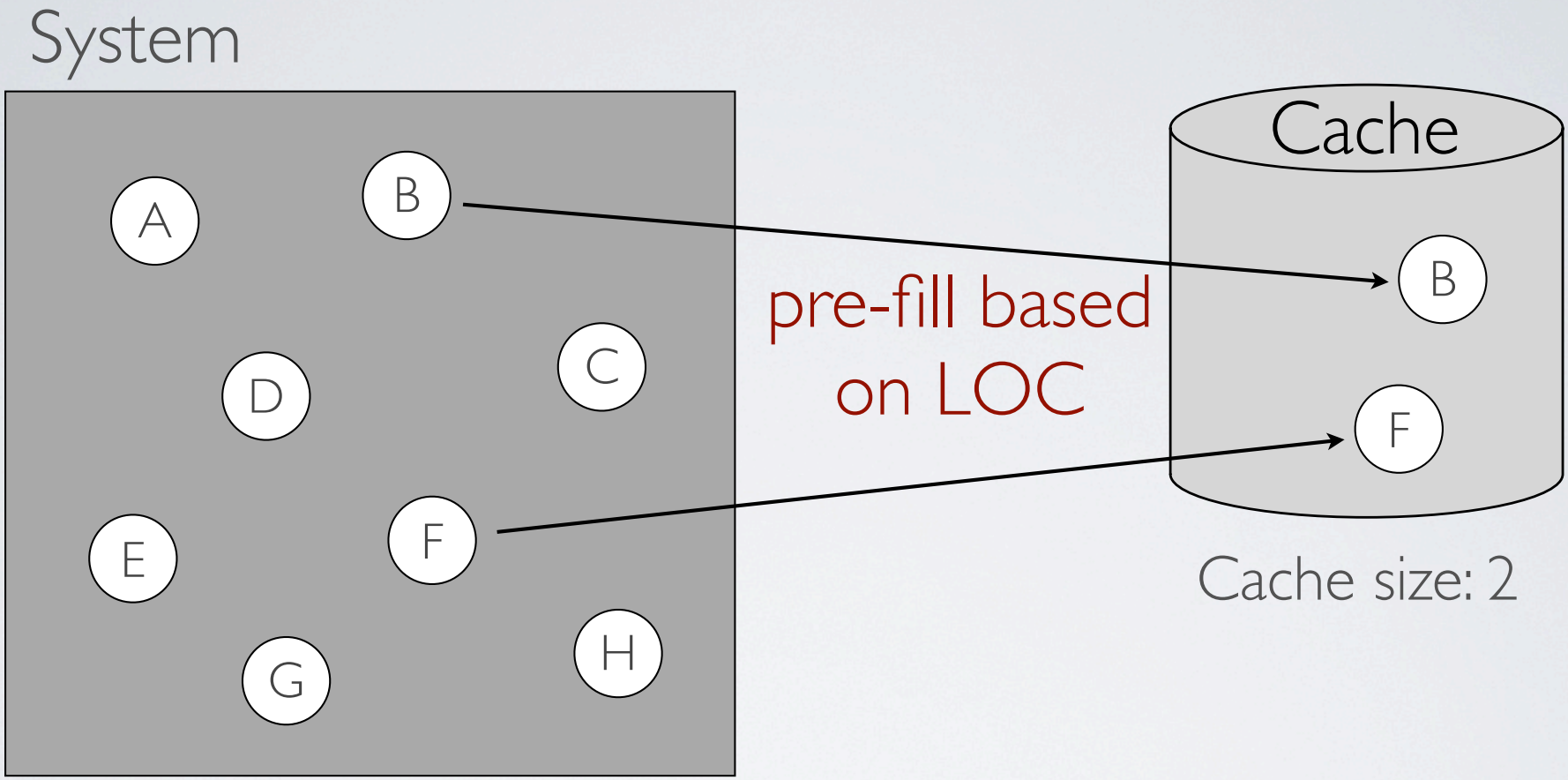
Kim et al., *Predicting faults from cached history,* ICSE 2007

Predicting defects with BugCache

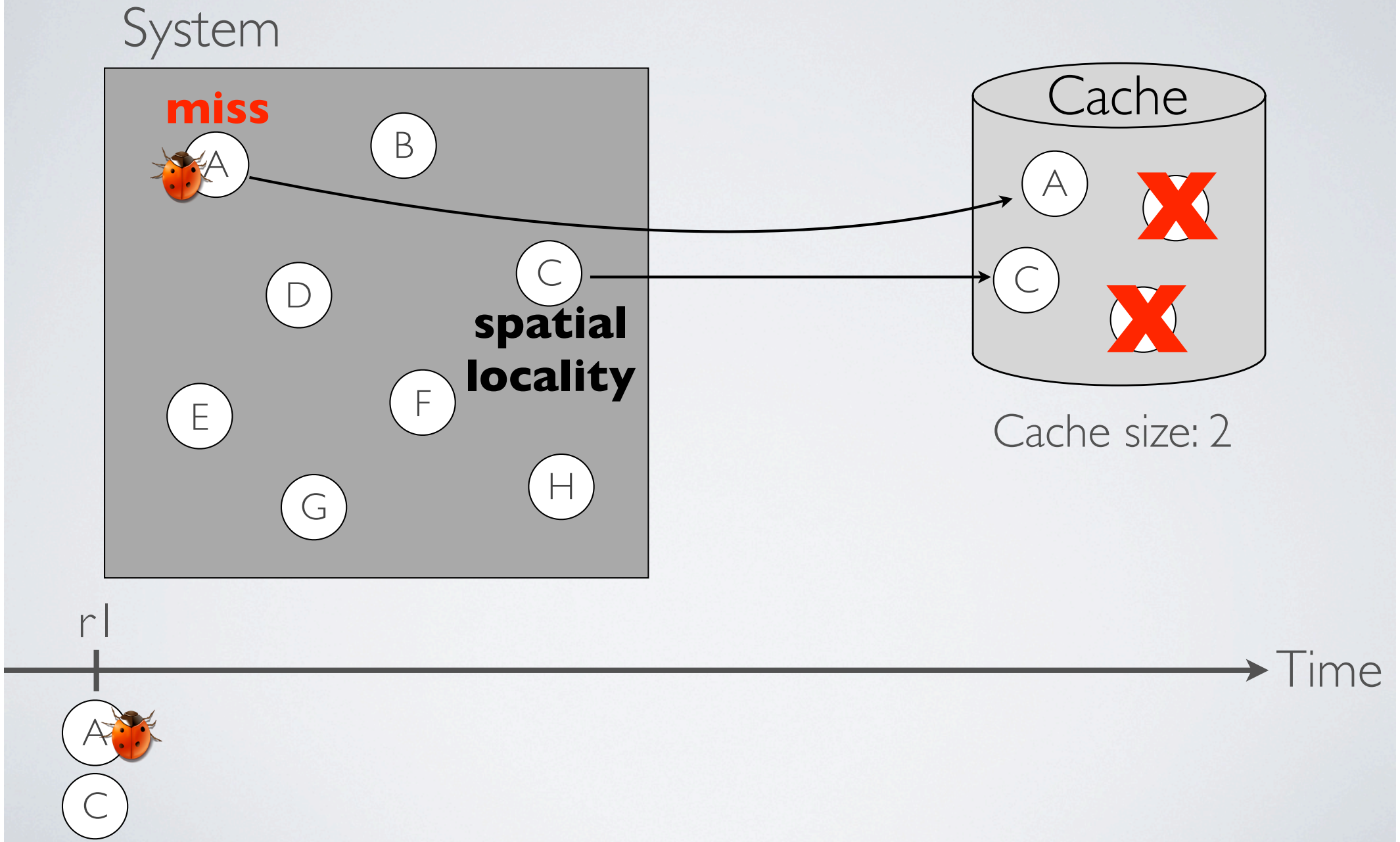




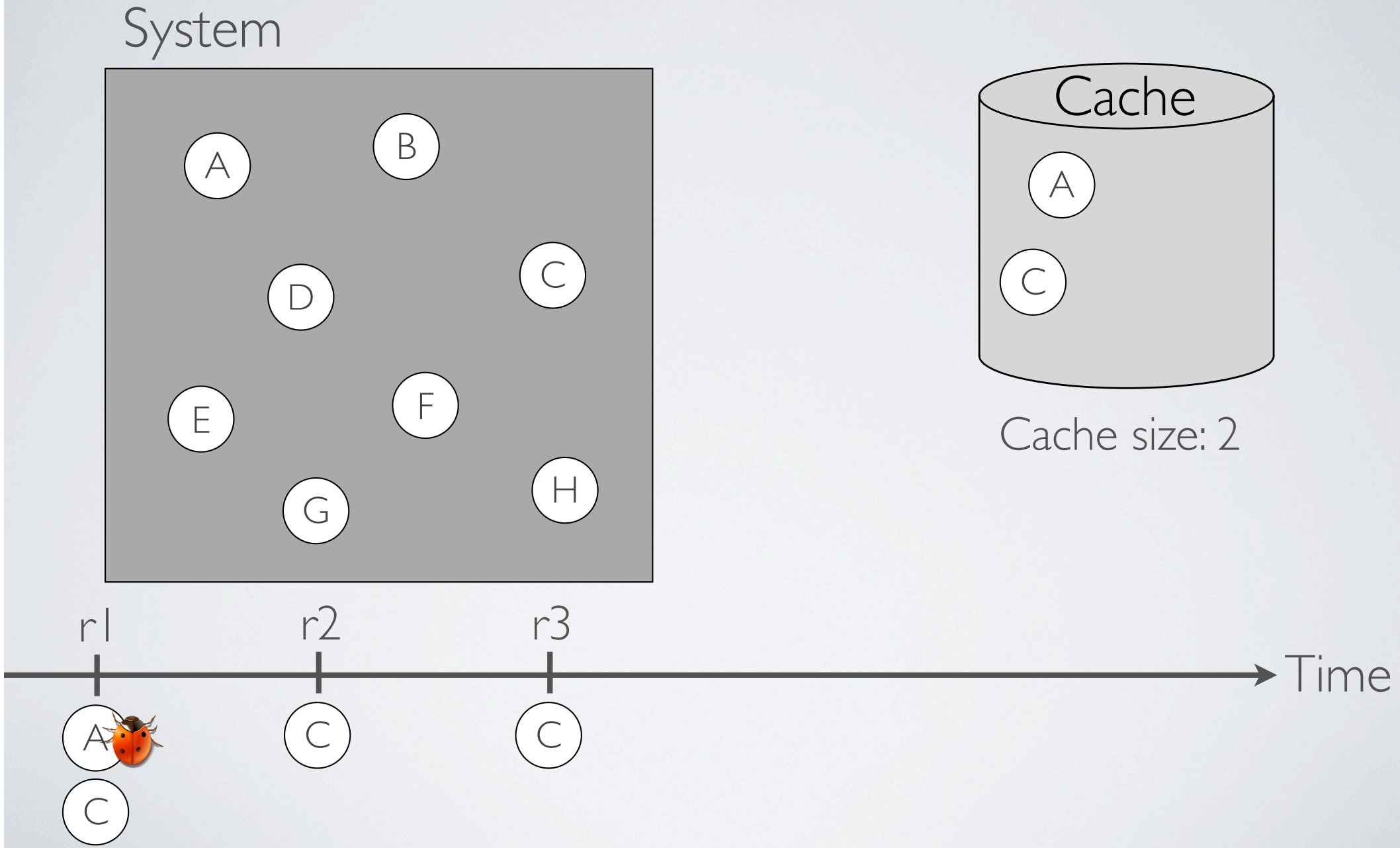
Predicting defects with BugCache



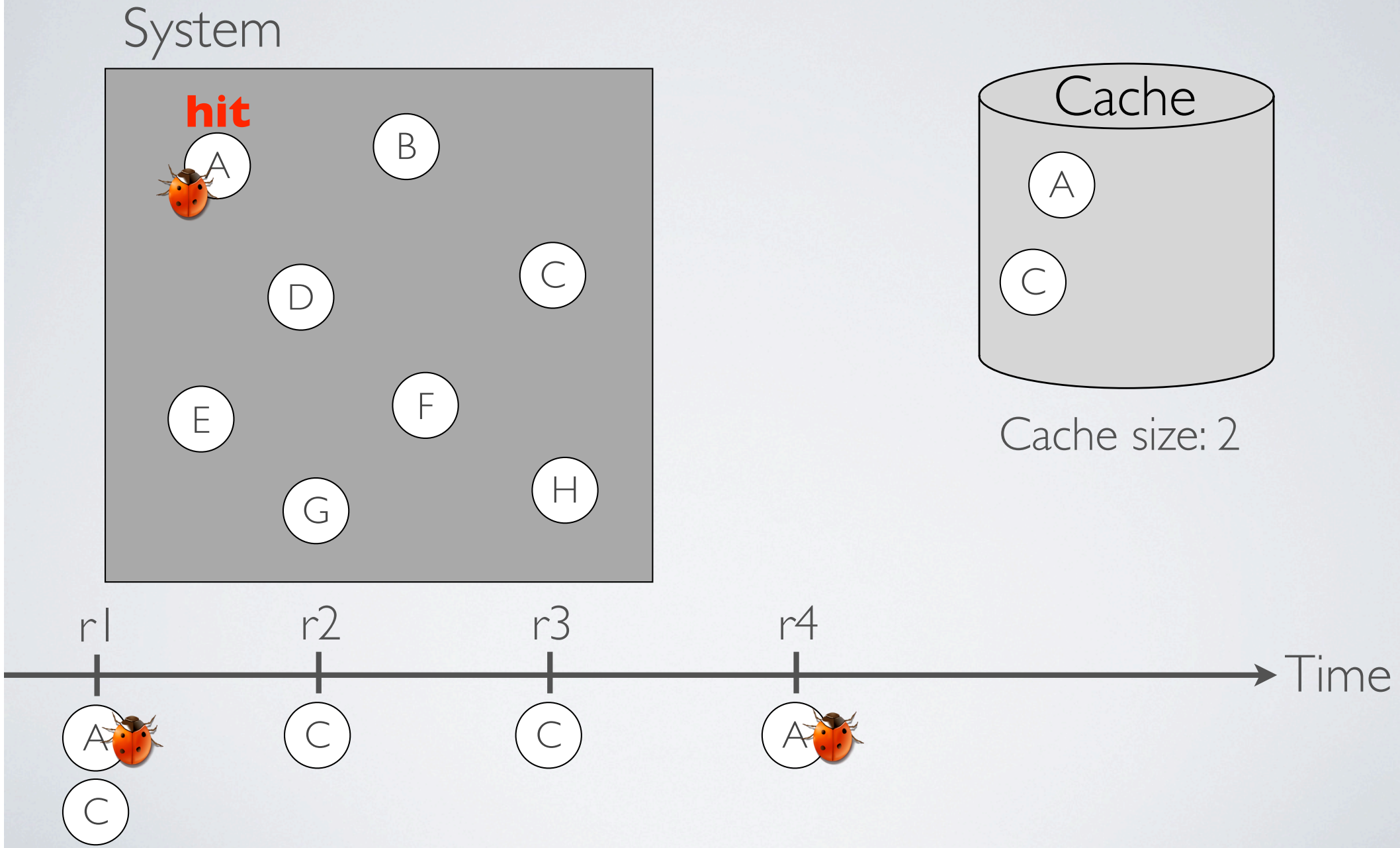
Predicting defects with BugCache



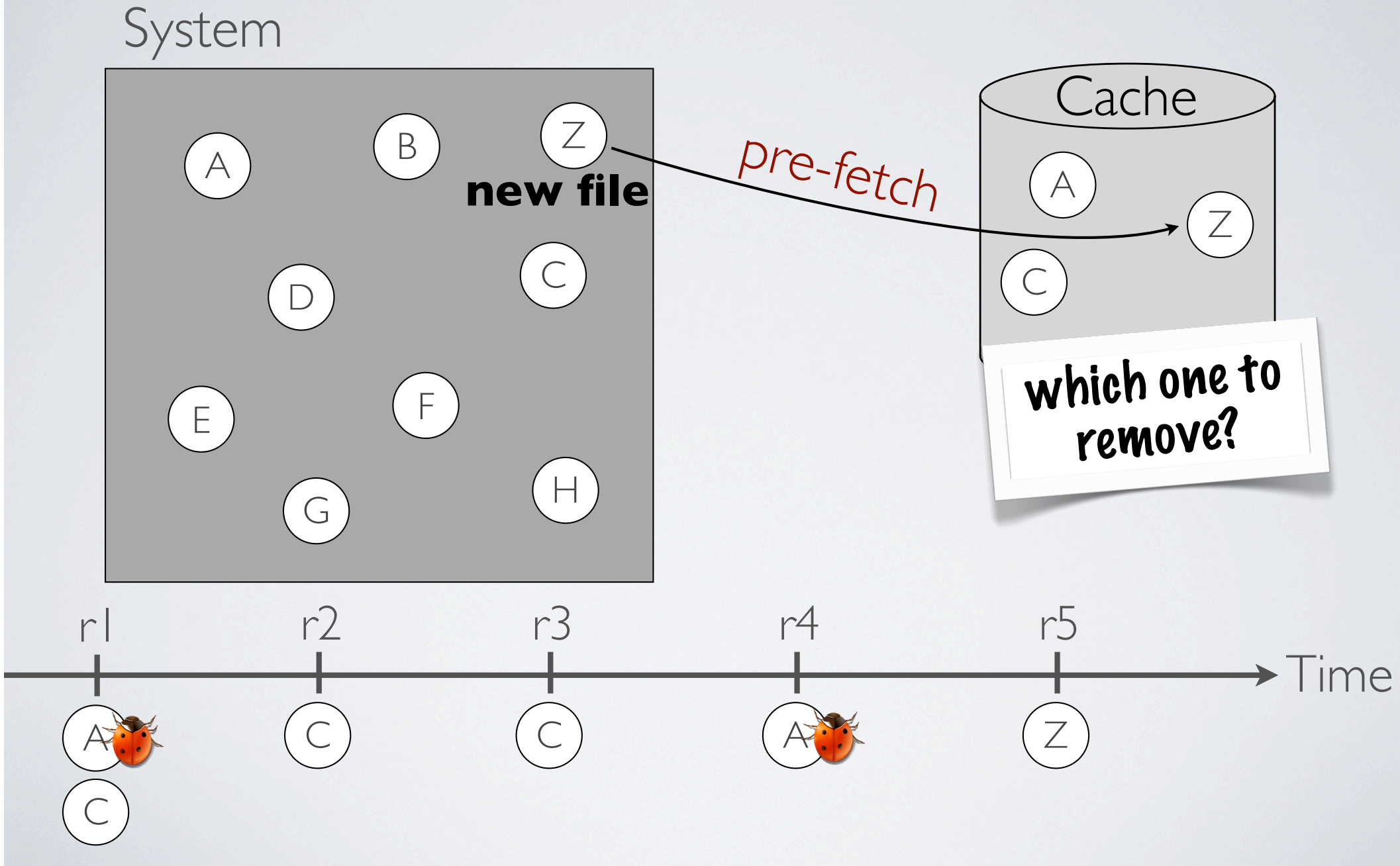
Predicting defects with BugCache



Predicting defects with BugCache

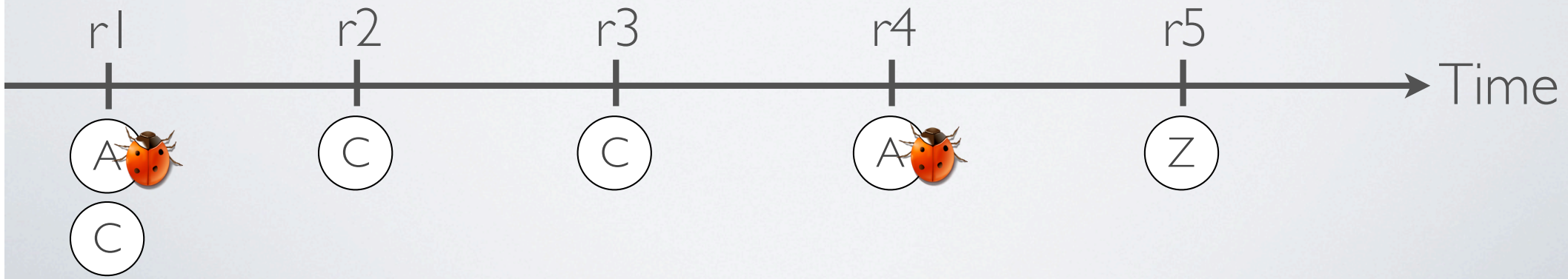


Predicting defects with BugCache



Replacement policy

- Least recently used: the files that have the least recently found defects (C)
- Least frequently changed (A)
- Least frequent defects (C)



Updating the cache

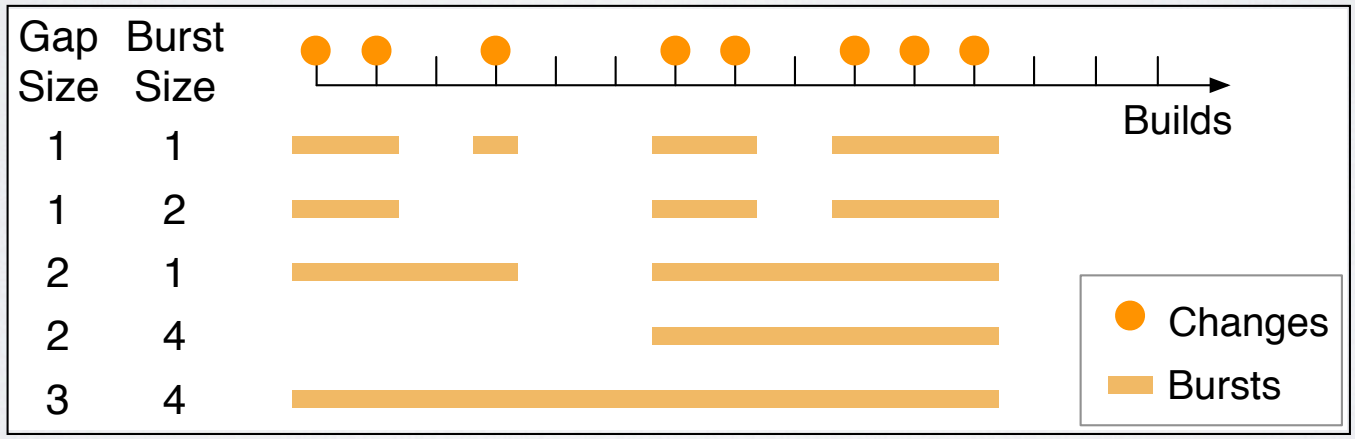
- At every iteration, the cache is updated based on:
  - misses: files which exhibit bugs are added to the cache
  - pre-fetching:
    - adding changed files
    - adding new files
    - removing deleted files

Evaluation

- BugCache evaluated on 7 open source projects
- Cache size: 10% of the system files
- This 10% files account for 73% - 95% of all defects
- Previous work, with 10% of files, covered up to 78%

Predicting defects with change burst

- Change burst is one of the best performing approaches
- The hypothesis is that "burst" of changes are very likely to introduce defects



Nagappan et al., *Change Bursts as Defect Predictors*, ISSRE 2010

Change burst performances on Windows Vista

	Precision	Recall
Pre-Release Defects	73.8%	62.9%
Test coverage	83.8%	54.4%
Dependencies	74.4%	69.9%
Code complexity	79.3%	66.0%
Code churn	78.6%	79.9%
Organizational structure	86.2%	84.0%
Change burst	**91.1%**	**92.0%**

Performances on other systems

- On Windows Server 2003 change burst is among the best predictors
  - precision 74.4% - recall 88.0%
- On Eclipse 2.0 change burst has the same performance as change metrics
  - precision 67.0% - recall 51.0%

why so "bad"?

- Change burst needs a controlled change process to be applied
  - A process in which changes are committed only when expected to keep the product stable
  - In Eclipse, instead, anyone can commit any change at any time

Prediction performances on Windows Vista

	Precision	Recall
Pre-Release Defects	73.8%	62.9%
Test coverage	83.8%	54.4%
Dependencies	74.4%	69.9%
Code complexity	79.3%	66.0%
Code churn	78.6%	79.9%
Organizational structure	86.2%	84.0%
Change burst	91.1%	92.0%

Prediction performances on Windows Vista

	Precision	Recall
Pre-Release Defects	73.8%	62.9%
Test coverage	83.8%	54.4%
Dependencies		69.9%
Code complexity		66.0%
Code churn	78.6%	79.9%
Organizational structure	86.2%	84.0%
Change burst	91.1%	92.0%

Defects are not all the same!

Defects are not all the same

- Different defects have different cost
- If the goal of defect prediction is to optimize QA resources, the cost of QA activity per defect should be taken into account
- The QA cost should be part of the prediction model
- Effort-aware defect prediction

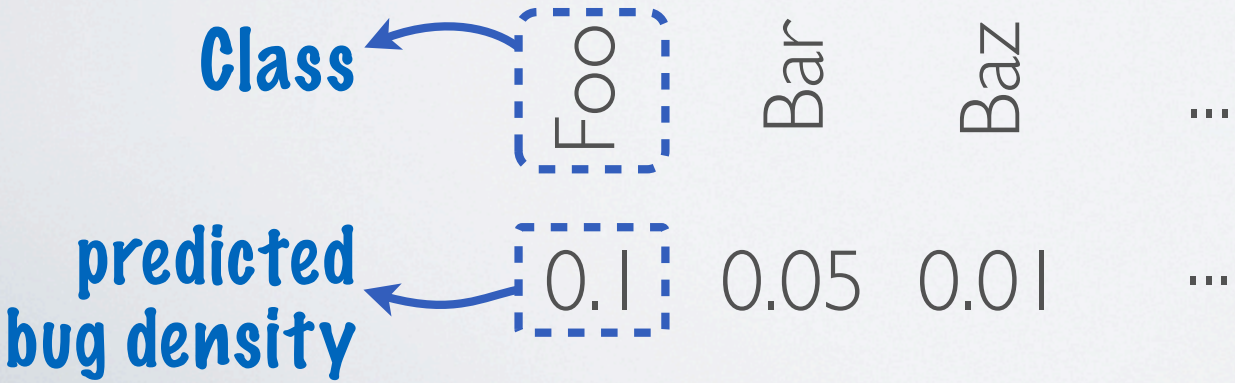
Effort-aware prediction performance

Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009

Foo	Bar	Baz	...
0.1	0.05	0.01	...

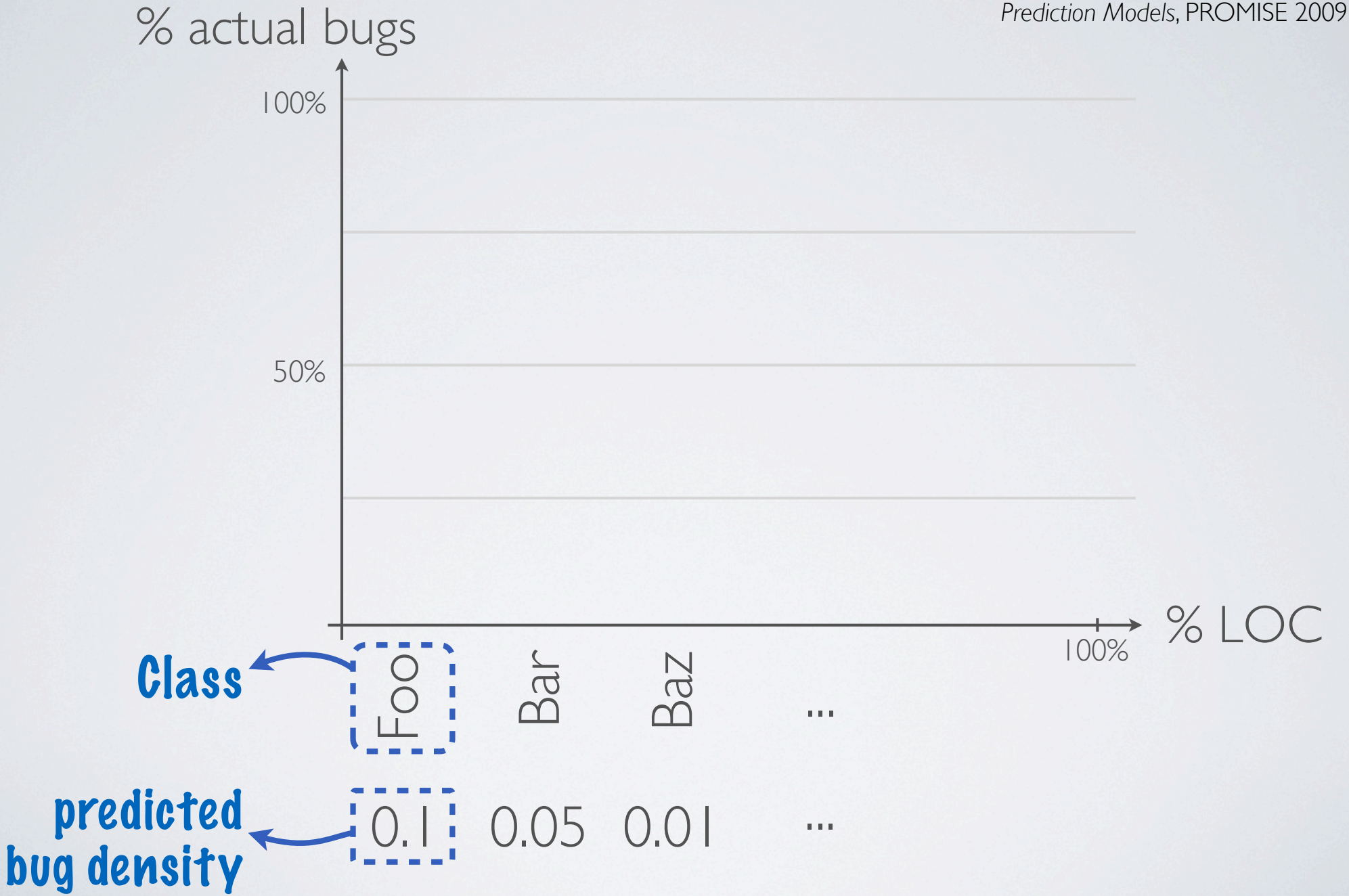
Effort-aware prediction performance

Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009



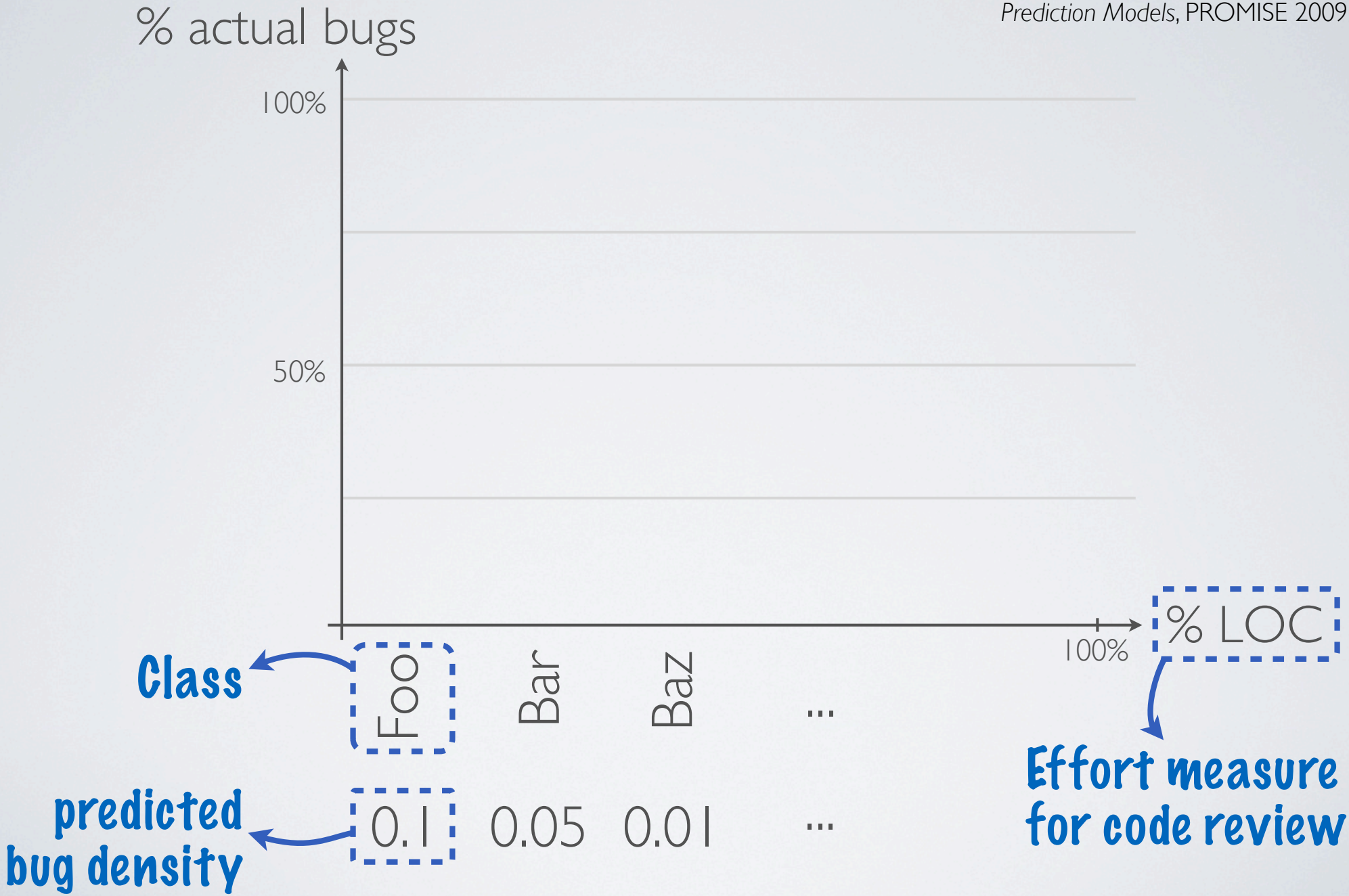
Effort-aware prediction performance

Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009



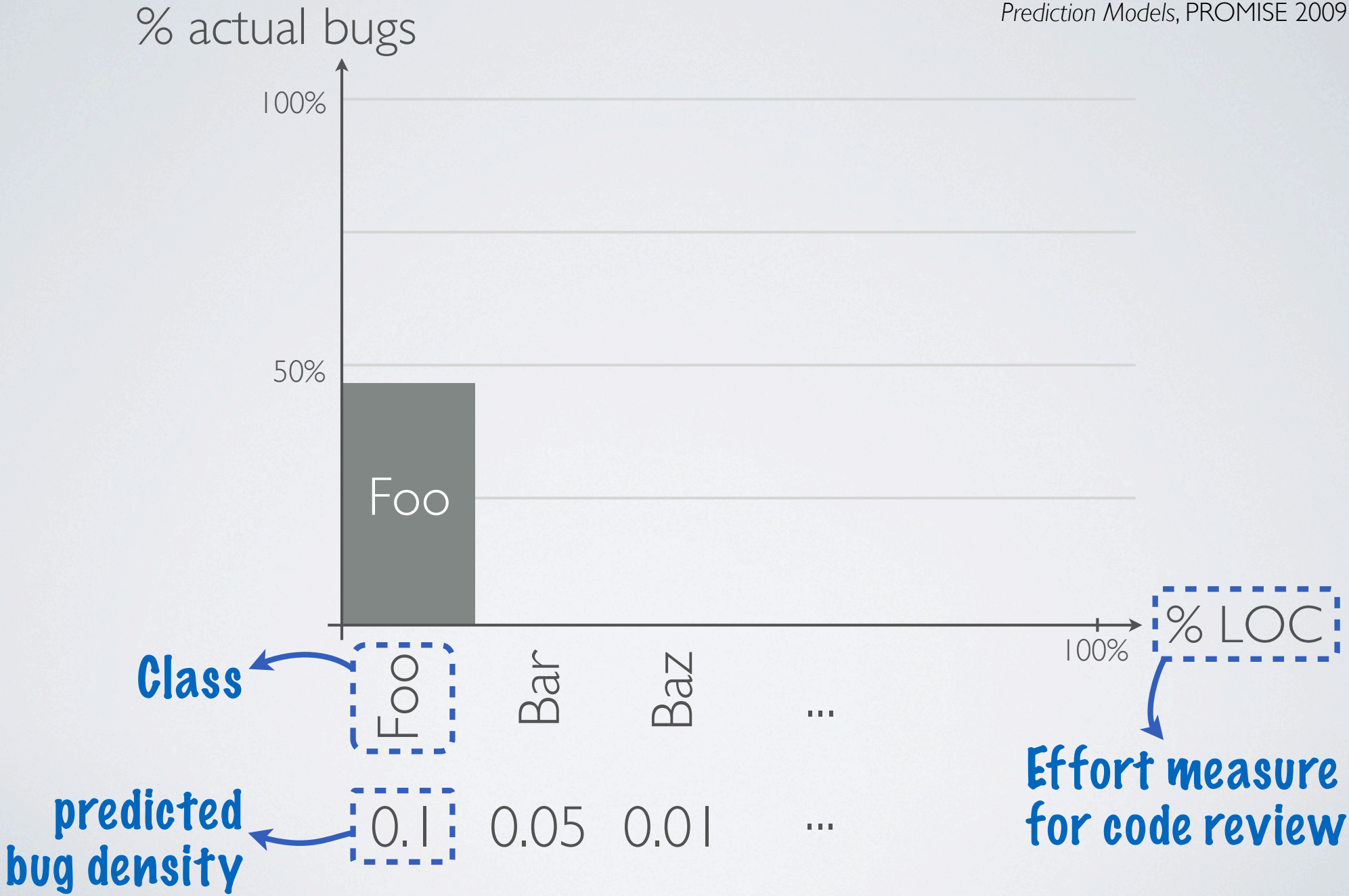
Effort-aware prediction performance

Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009



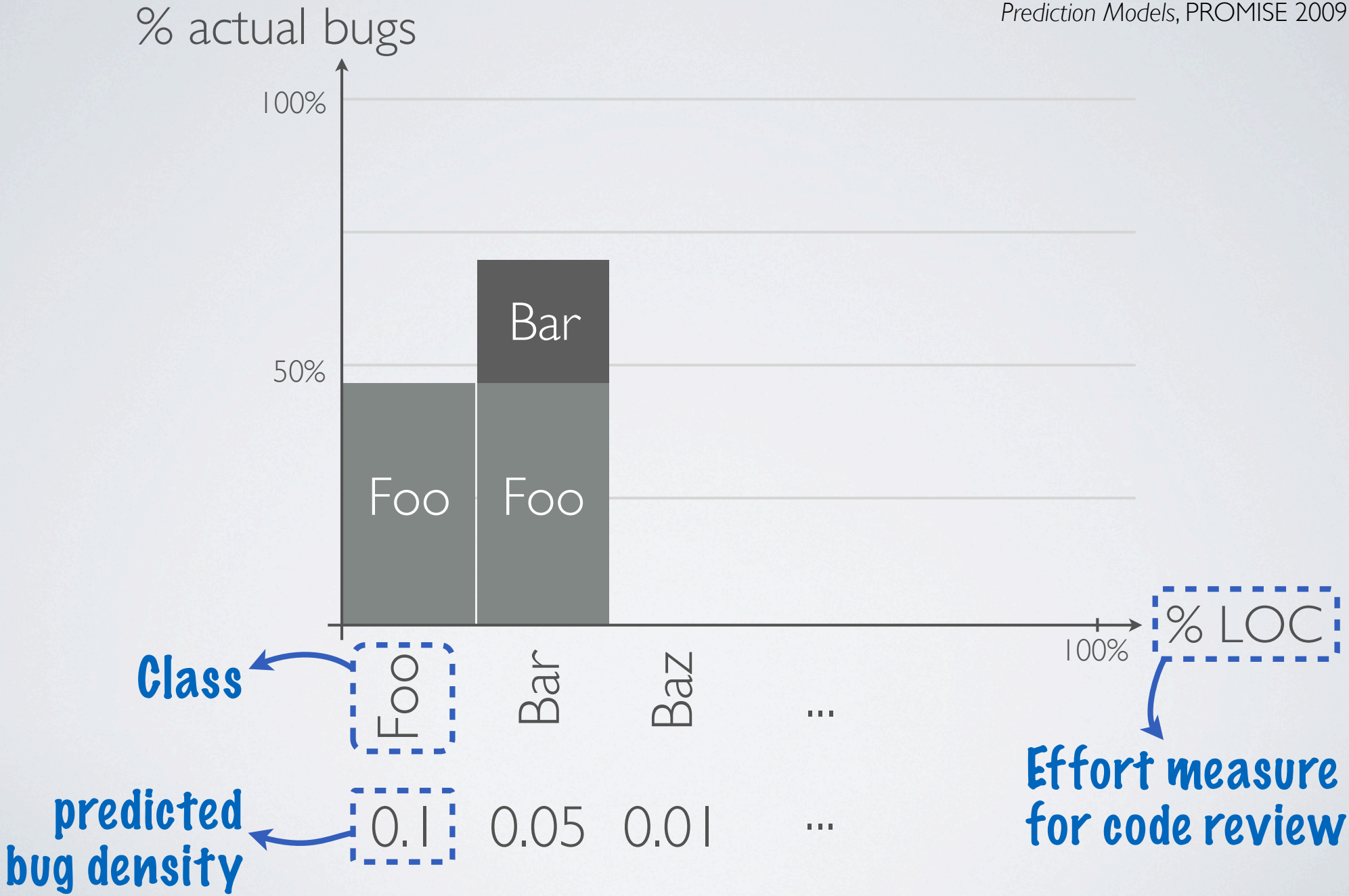
Effort-aware prediction performance

Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009



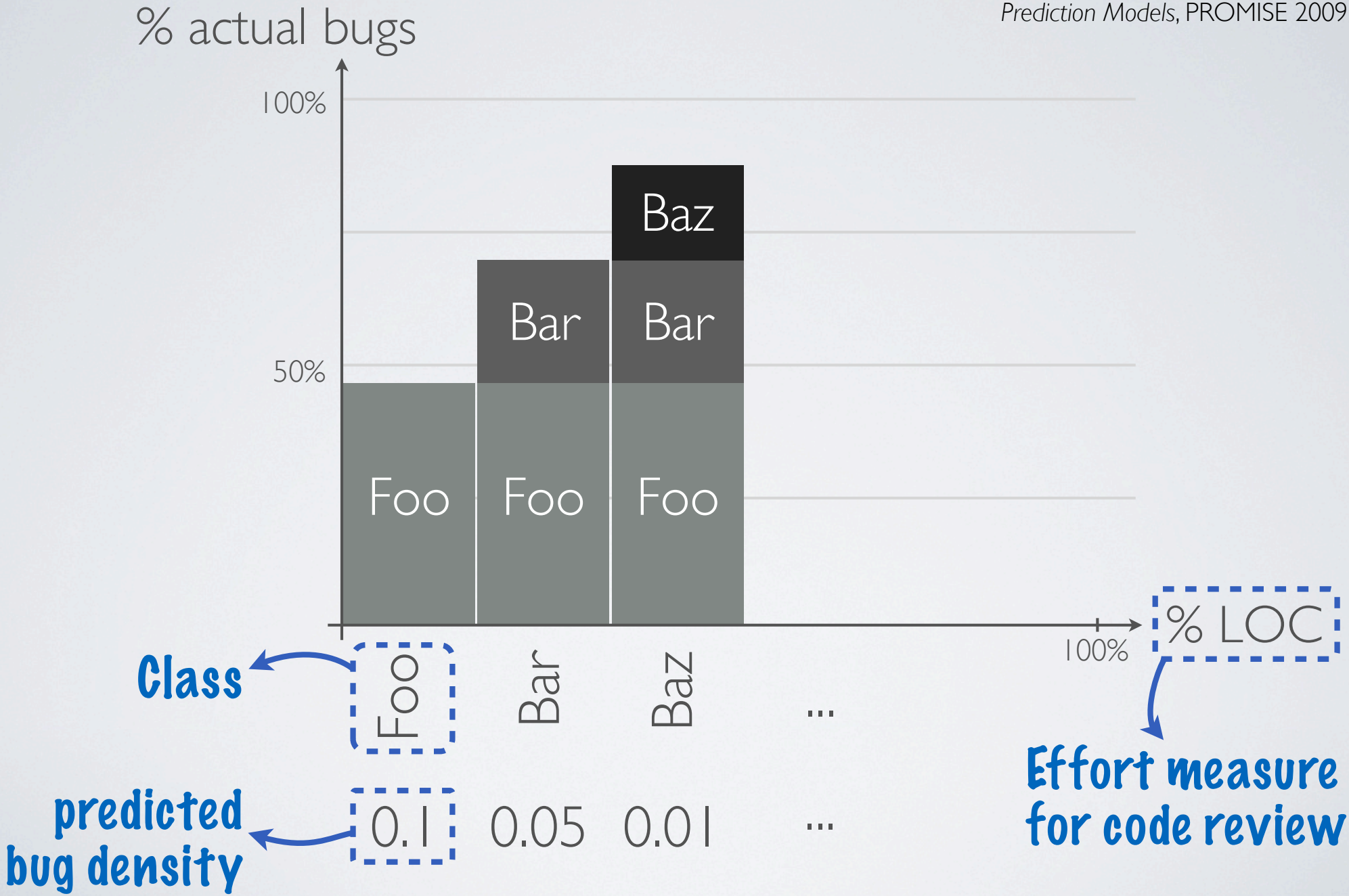
Effort-aware prediction performance

Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009



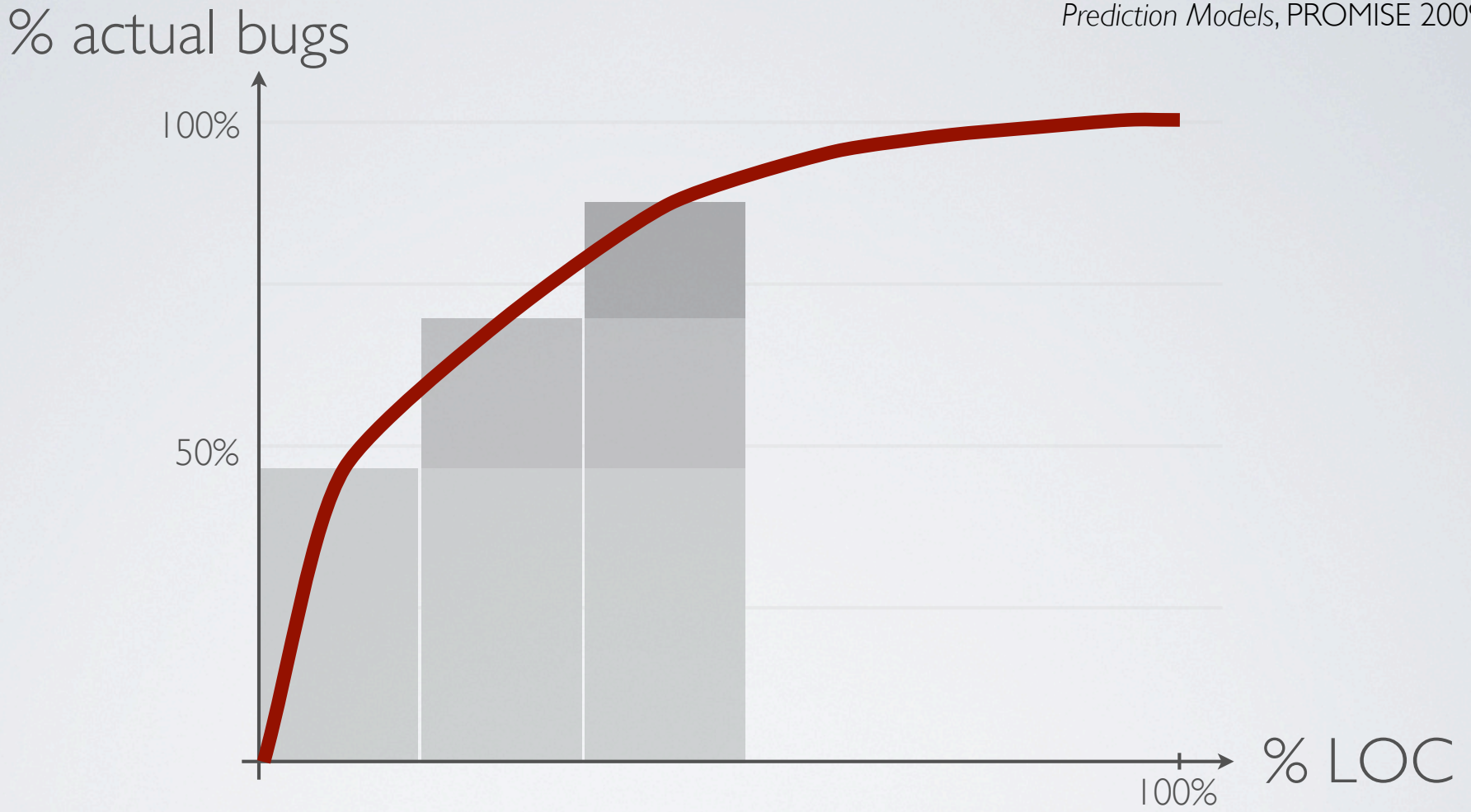
Effort-aware prediction performance

Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009

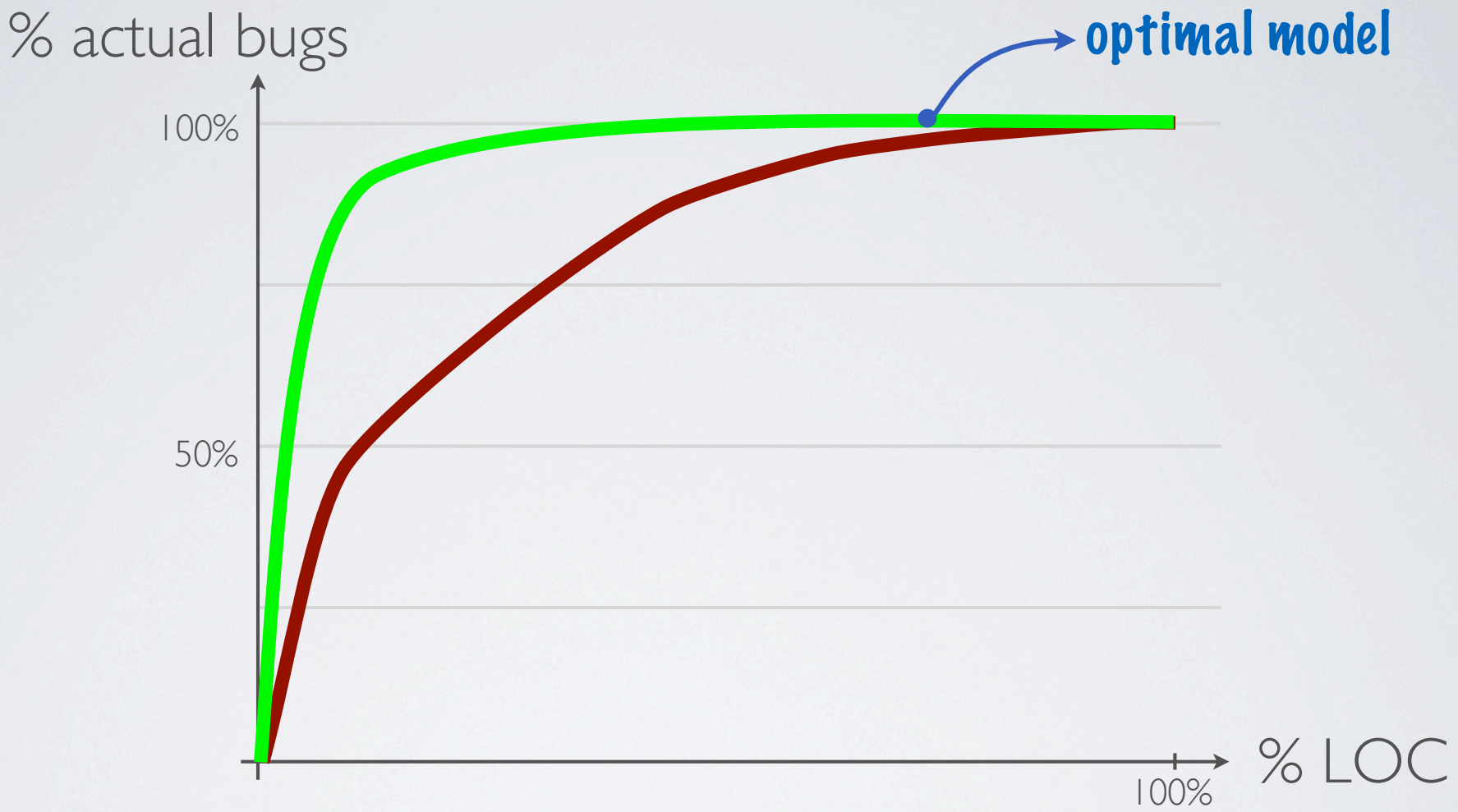


Effort-aware prediction performance

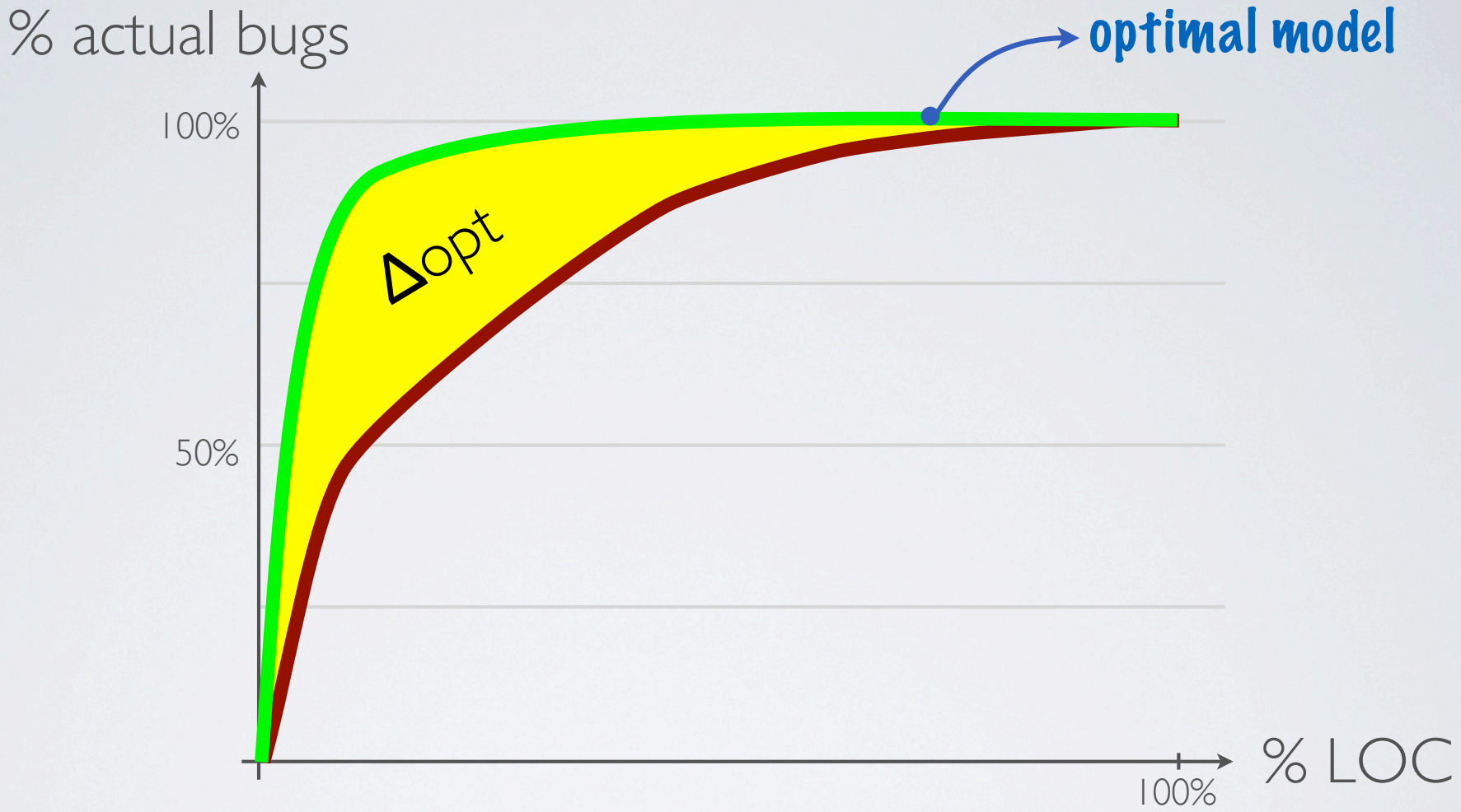
Mende and Koschke, *Revisiting the Evaluation of Defect Prediction Models*, PROMISE 2009



Effort-aware prediction performance



Effort-aware prediction performance



similar to AUC ⟵ Popt = 1 - Δopt

Comparison with effort-unaware measures

- Evaluation on 13 datasets from NASA
- Prediction models performing well when evaluated without the effort, have bad effort-aware performance
- All models are far from an optimal effort-aware prediction

Other work on effort-aware defect prediction

- Arisholm and Briand, *Predicting fault-prone components in a java legacy system,* ISESE 2006
- Koru et al., *Modeling the effect of size on defect proneness for open-source software*, PROMISE 2007
- Koru et al., *Theory of relative defect proneness*, Empirical Software Engineering 2008
- Arisholm et al., *A systematic and comprehensive investigation of methods to build and evaluate fault prediction models*, Journal of System and Software 2010
- Kamei et al., *Revisiting common bug prediction findings using effort aware models*, ICSM 2010
- Menzies et al., *Defect prediction from static code features: current results, limitations, new approaches*, Automated Software Engineering 2010

Defect prediction approaches are difficult to compare

- Applied on different systems
- Different granularity levels (file, class, module, package, binary, subsystem)
- Different evaluation performances

An extensive comparison of defect prediction approaches

- Different approaches based on code metrics, change metrics, previous defects, entropy of changes, entropy of code metrics, churn of code metrics
- Applied on 5 different systems
- Different scenarios (classification, ranking, effort-aware ranking) and different models (logistic/linear regression, decision trees, Naïve Bayes)
- The best performing approach varies from system to system, and from scenario to scenario:
  - It is very difficult to show that an approach is better than another one in a statistical significant manner (across different systems)

D'Ambros et al., *Evaluating defect prediction approaches: a benchmark and an extensive comparison*, EMSE 2011

In conclusion

- As statistical significance is hard, when not impossible, to obtain, one should fine-tune a defect predictor to the specific task at hand
- Benchmarks are crucial for defect prediction, as they allow the comparison of approaches
  - PROMISE benchmark
  - D'Ambros et al. benchmark

http://promisedata.org

PROMISE data » Data Repository

http://promisedata.org/?cat=11

ROJADIRECTA Apple Yahoo! Google Maps YouTube Wikipedia News (307) Popular Links Sanpaolo

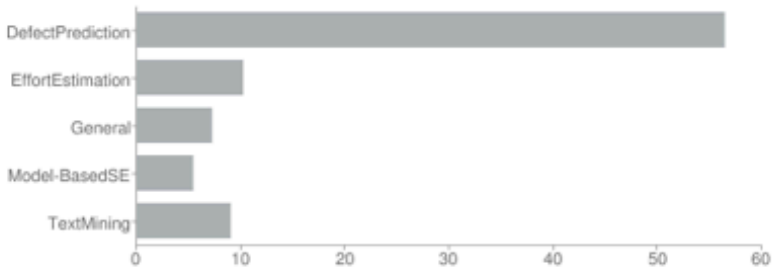
PROMISE 2011

140
139
138

Home | About | Data | Papers | People | PROMISE '06 | PROMISE '07 | PROMISE '08 | PROMISE '09 | PROMISE '10 | PROMISE '11

Search

Welcome to the Promise Data Repository!



In 2006, the repository held 23 data sets.

In 2008, at last update, the repository holds 141 data sets in the following areas:

- **Defect Prediction (96)**
- **Effort Estimation (for Bug Fixes) (1)**
- **Effort Prediction (18)**
- **General (9)**
- **Model-based SE (8)**
- **Text Mining (10)**

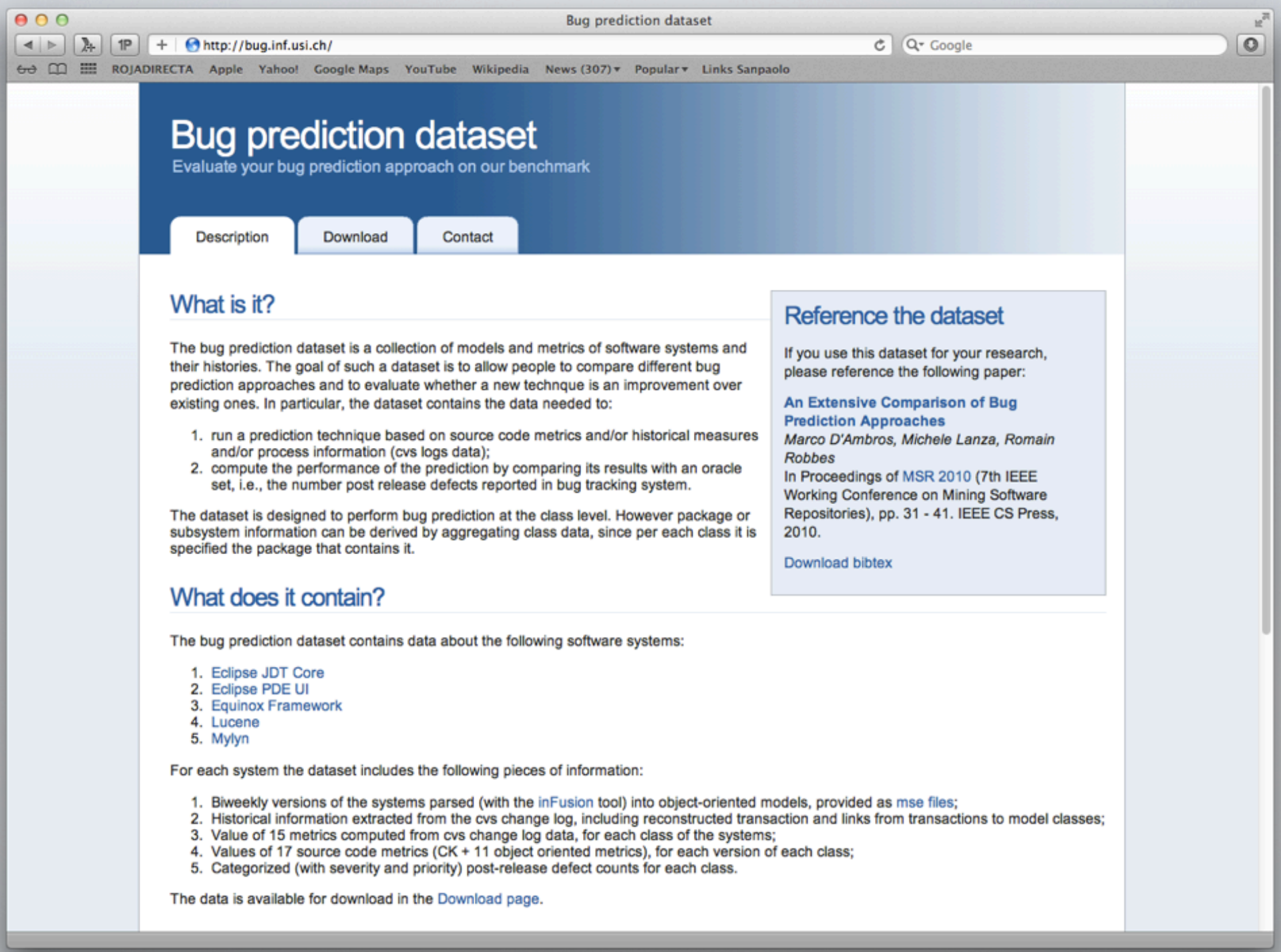
Further contributions are always welcome in the above areas, or any other.

Why so much data? Firstly, there is the open source effect: public code and public logs means more data sets.

News

- All presentations online
- Pictures – PROMISE 2008
- Promise Presentations Online
- Workshop Updates
- 2008 Workshop Updated

http://bug.inf.usi.ch/



Defect prediction

- Goal: Optimize available QA resources by
  - reducing code review cost
  - identifying defect-prone components
  - ranking component according to defect proneness

Defect prediction

Change analysis and prediction

Empirical studies

Expertise & bug triaging

Visual evolution analysis

Human factors

Change analysis and prediction

- Goal: Analyze software repository data to:
  - support change propagation and predict future changes
  - detect design problems (e.g. architecture decay)

Typical change prediction scenario

Experienced developer

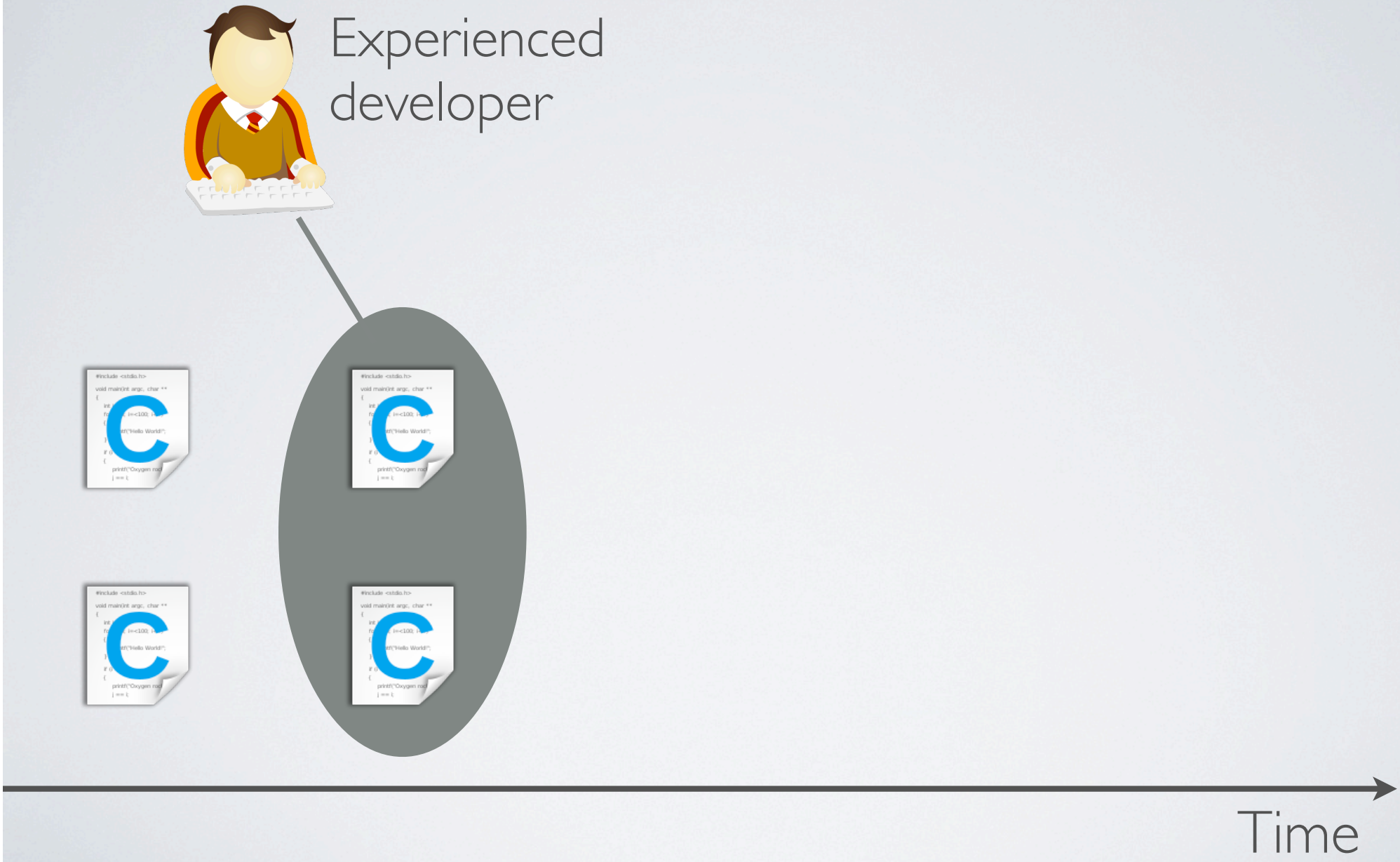
Time

Typical change prediction scenario

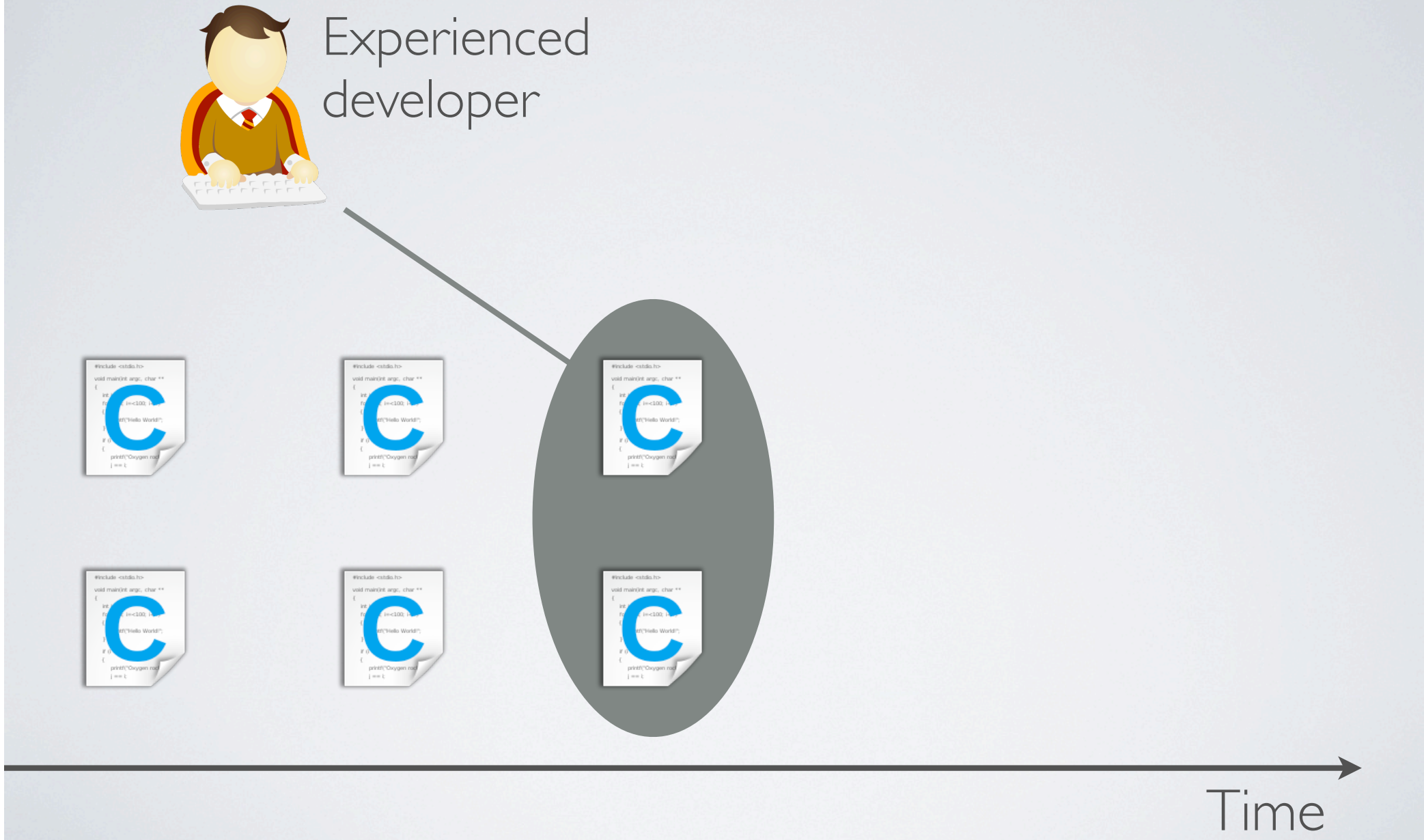
Experienced
developer

Time

Typical change prediction scenario



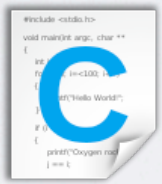
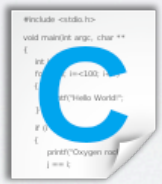
Typical change prediction scenario

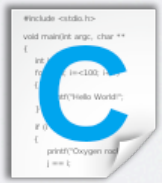


Typical change prediction scenario

Experienced developer

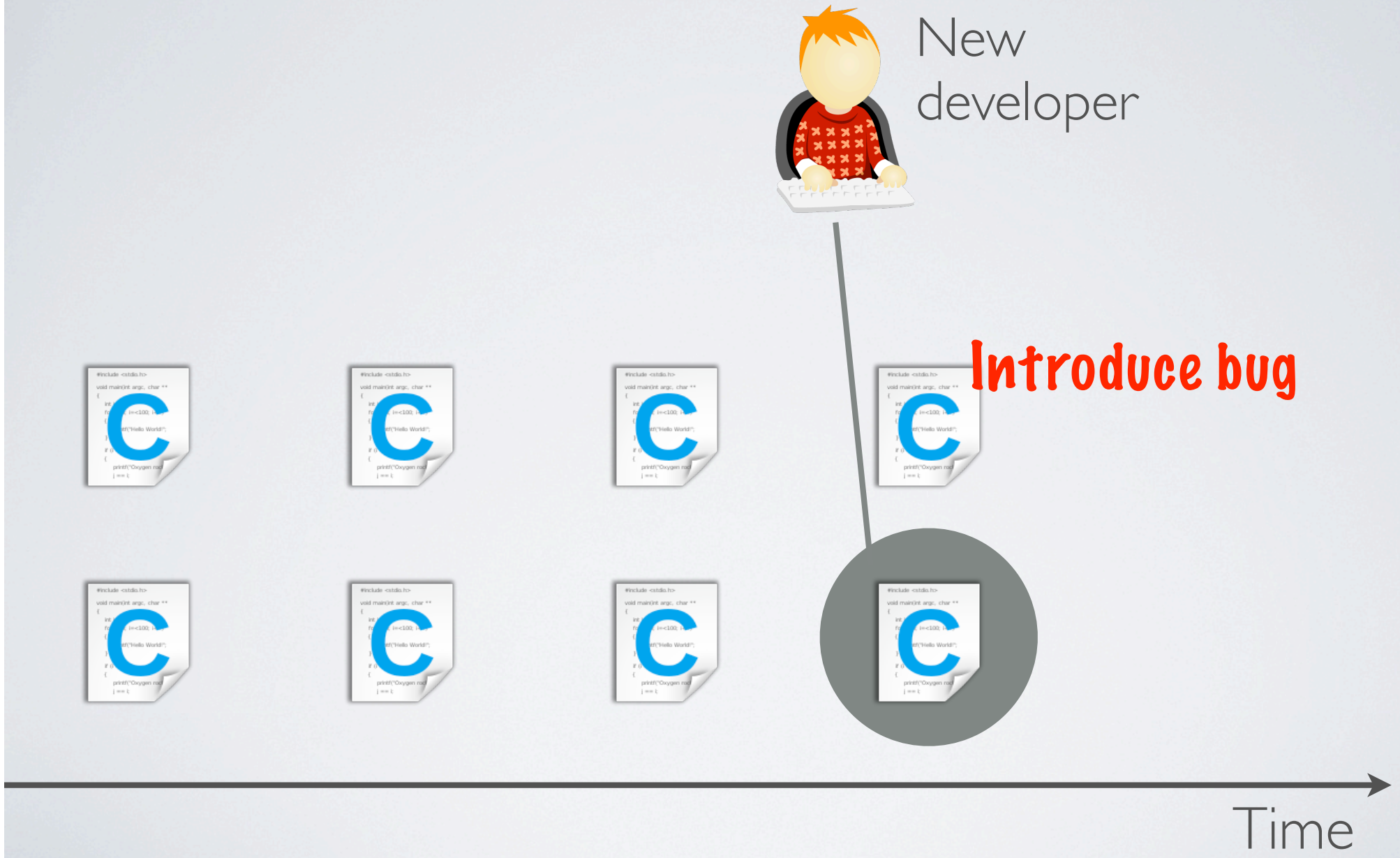
New developer

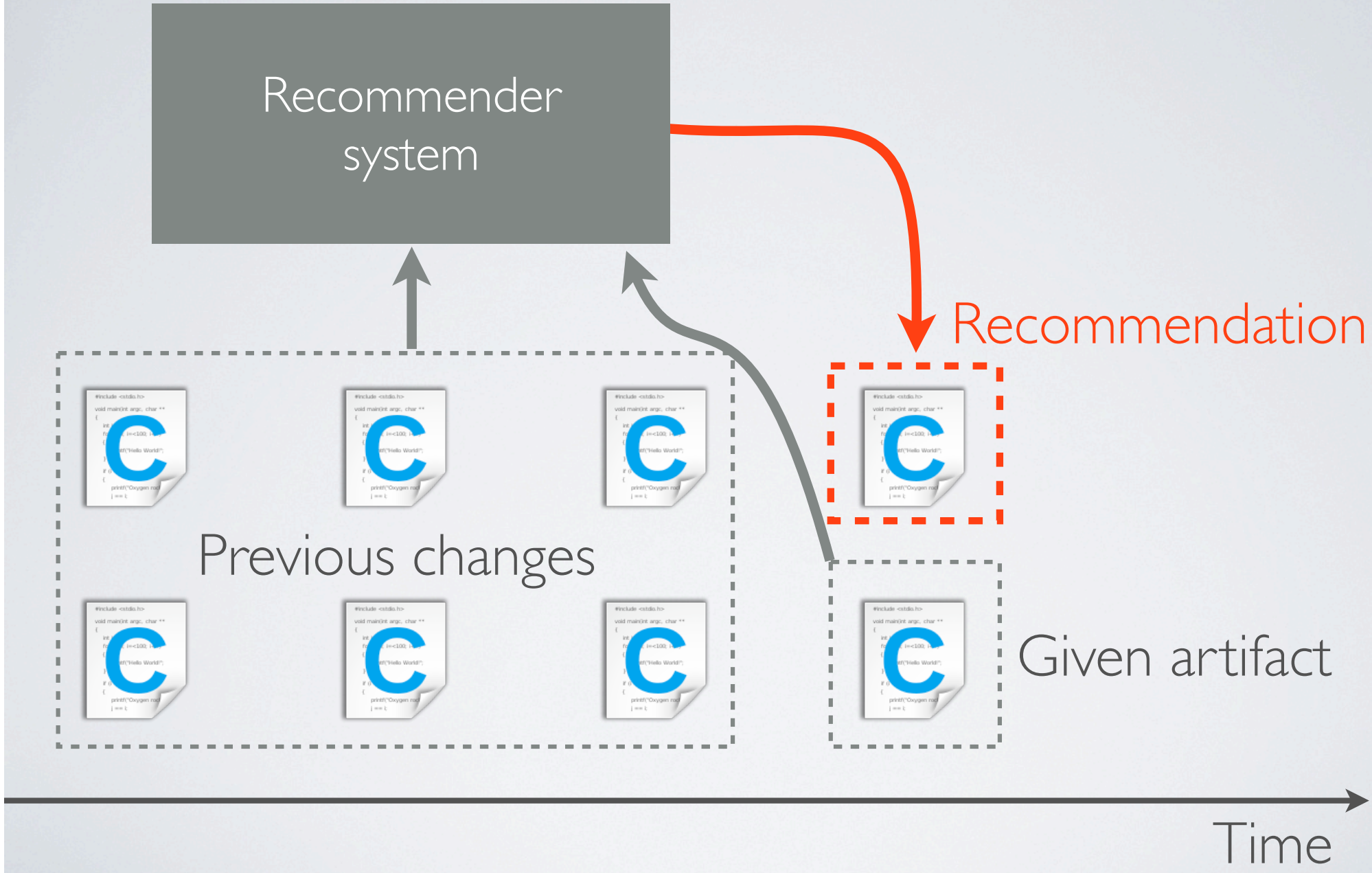



Time

Typical change prediction scenario

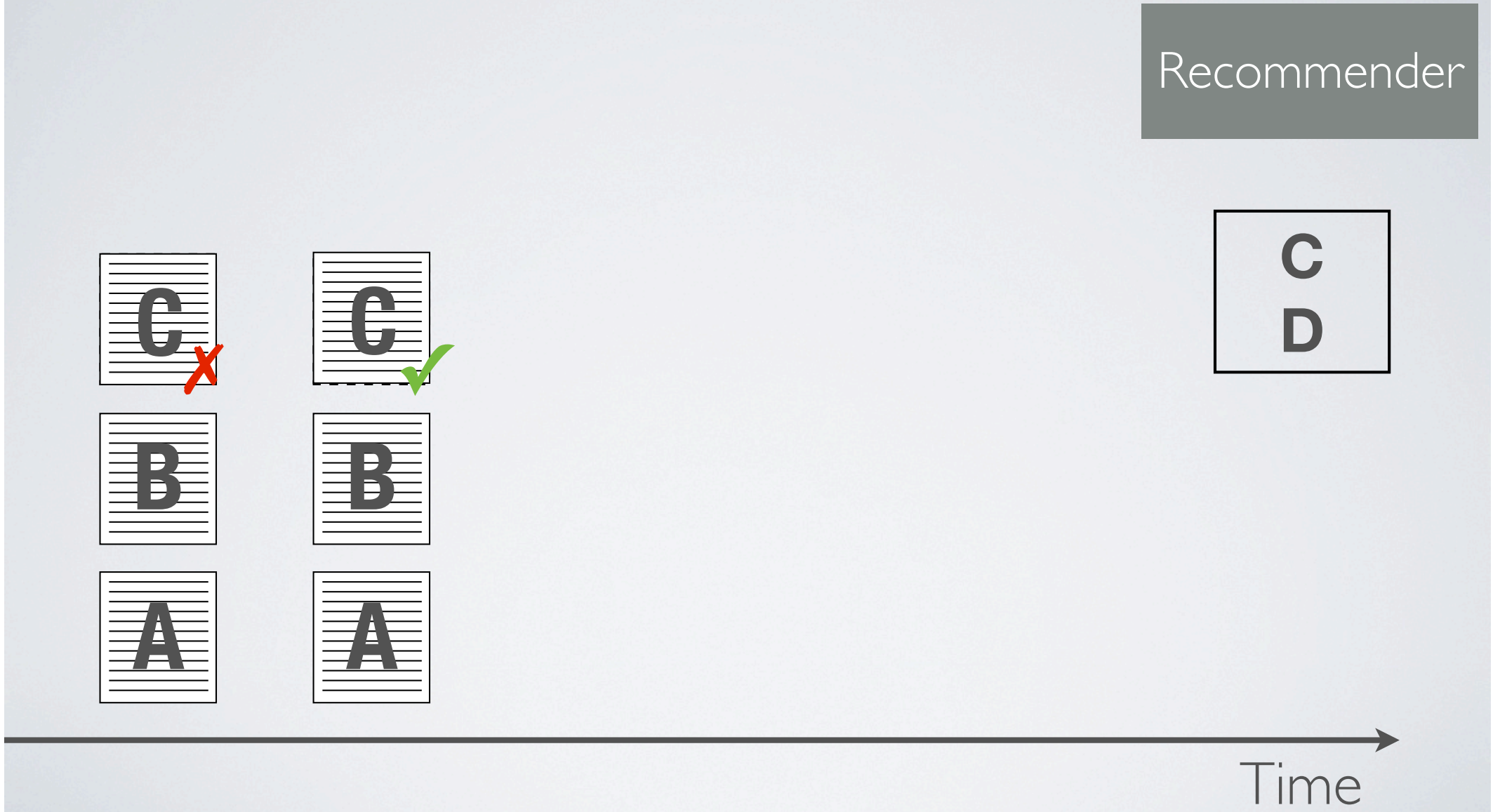


Typical change prediction scenario

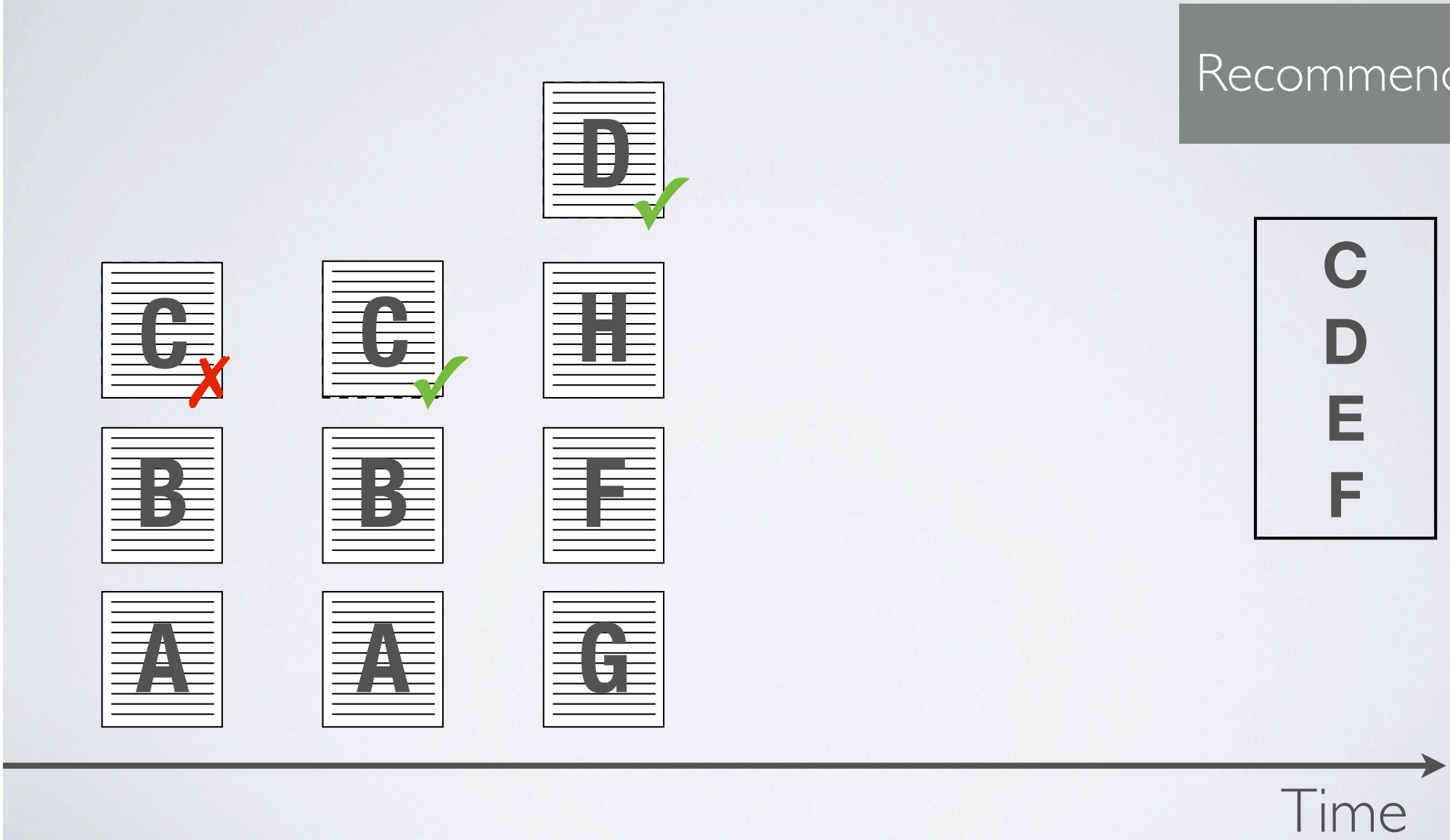


Evaluation of a change predictor through development replay

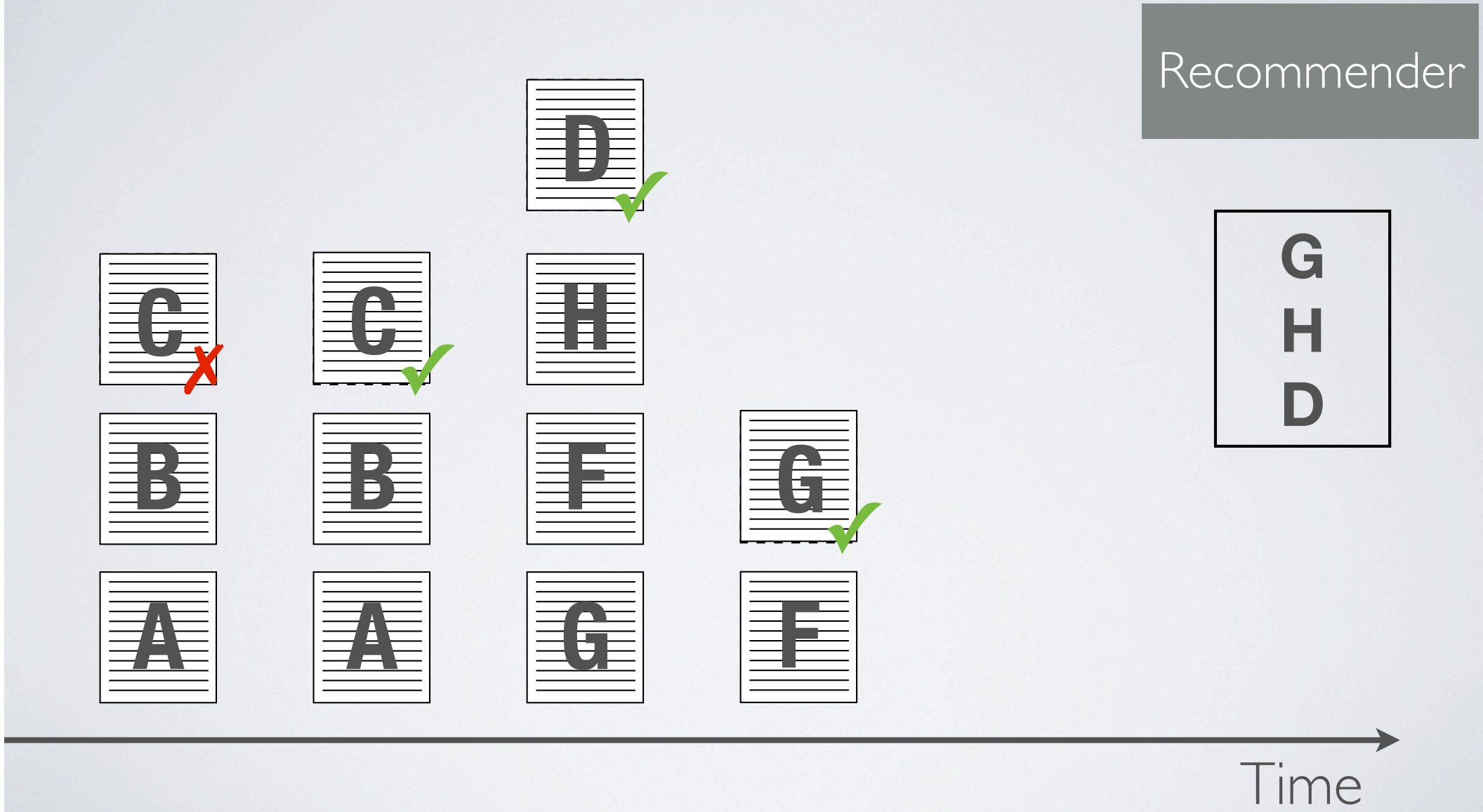
Evaluation of a change predictor through development replay



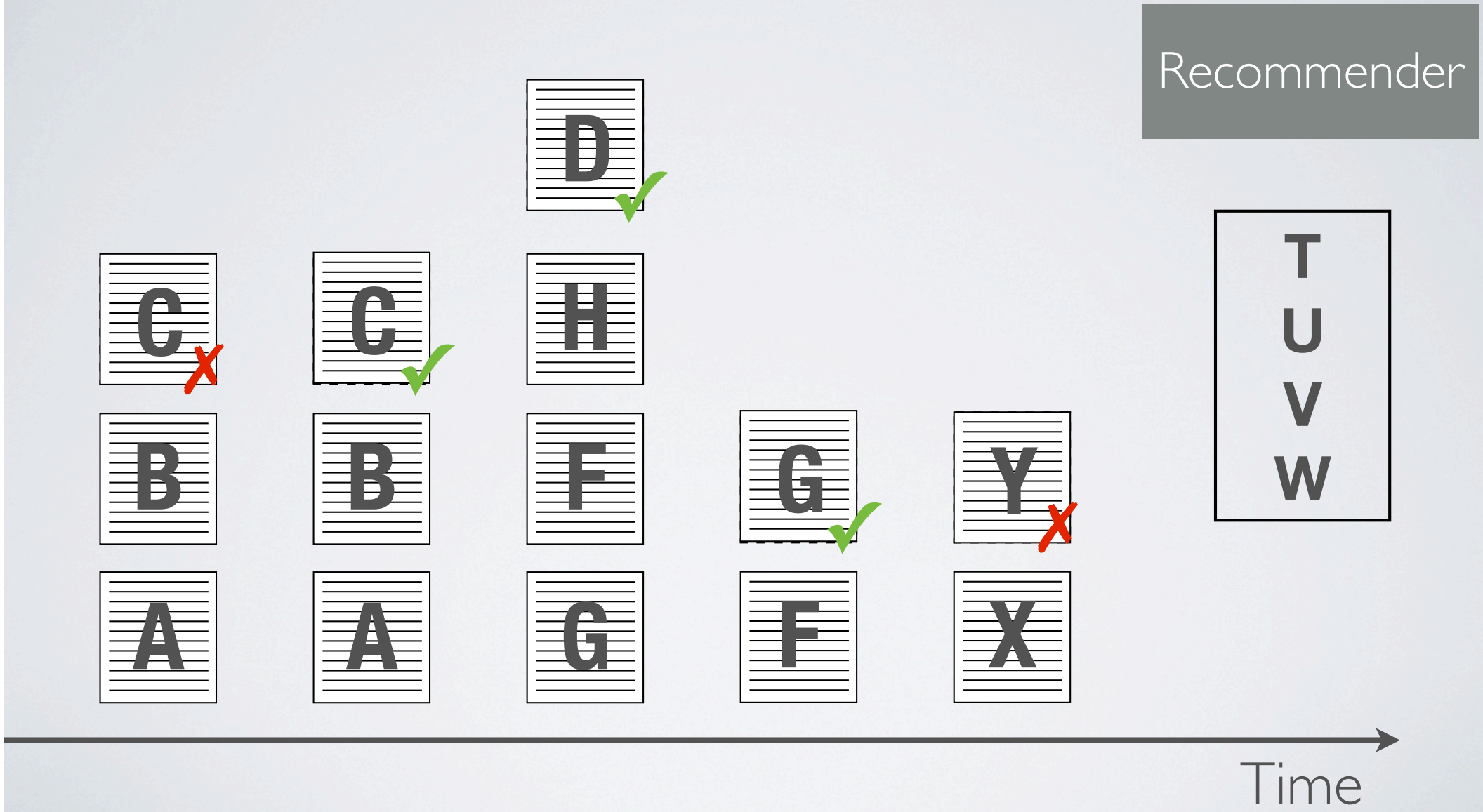
Evaluation of a change predictor through development replay



Evaluation of a change predictor through development replay

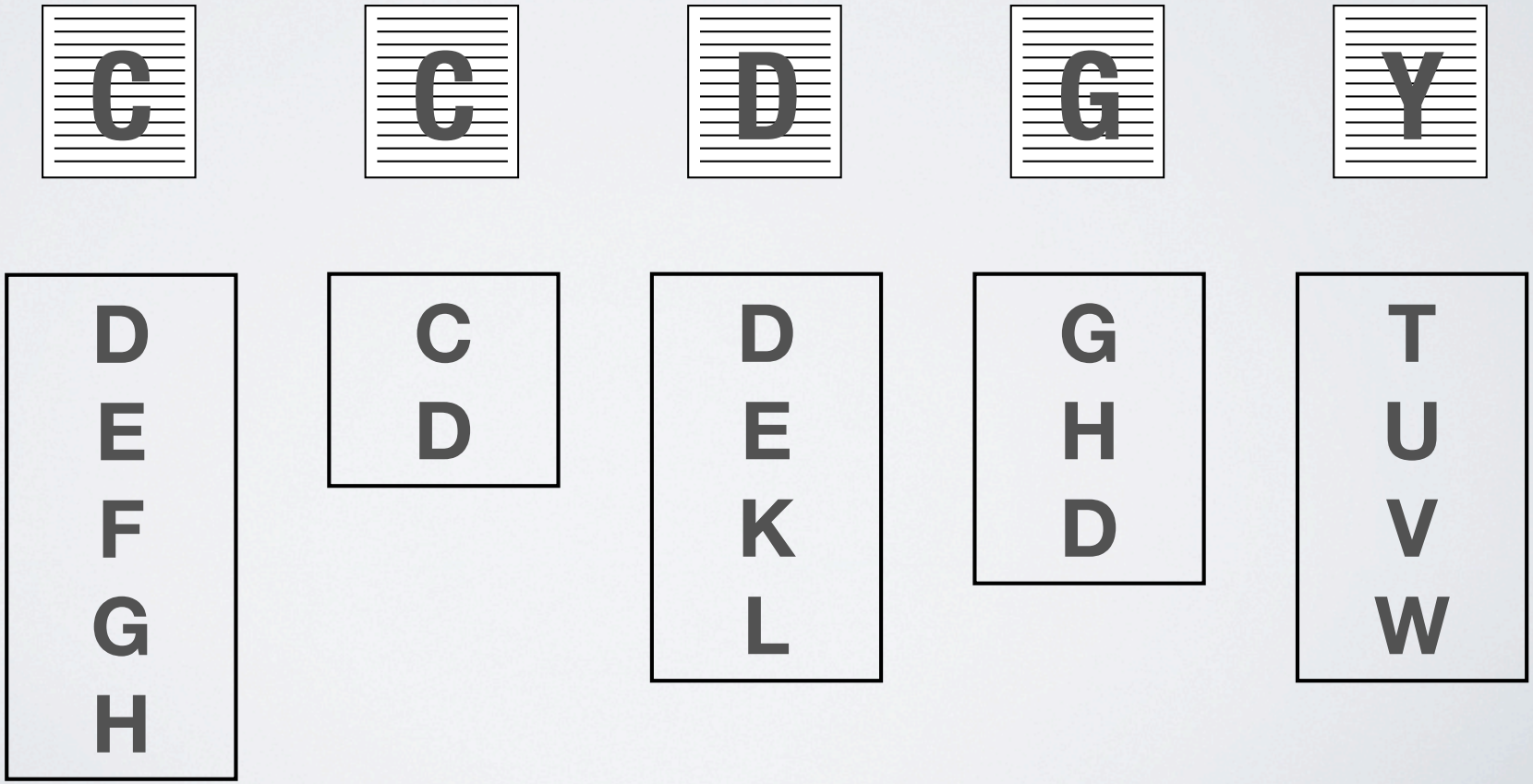


Evaluation of a change predictor through development replay

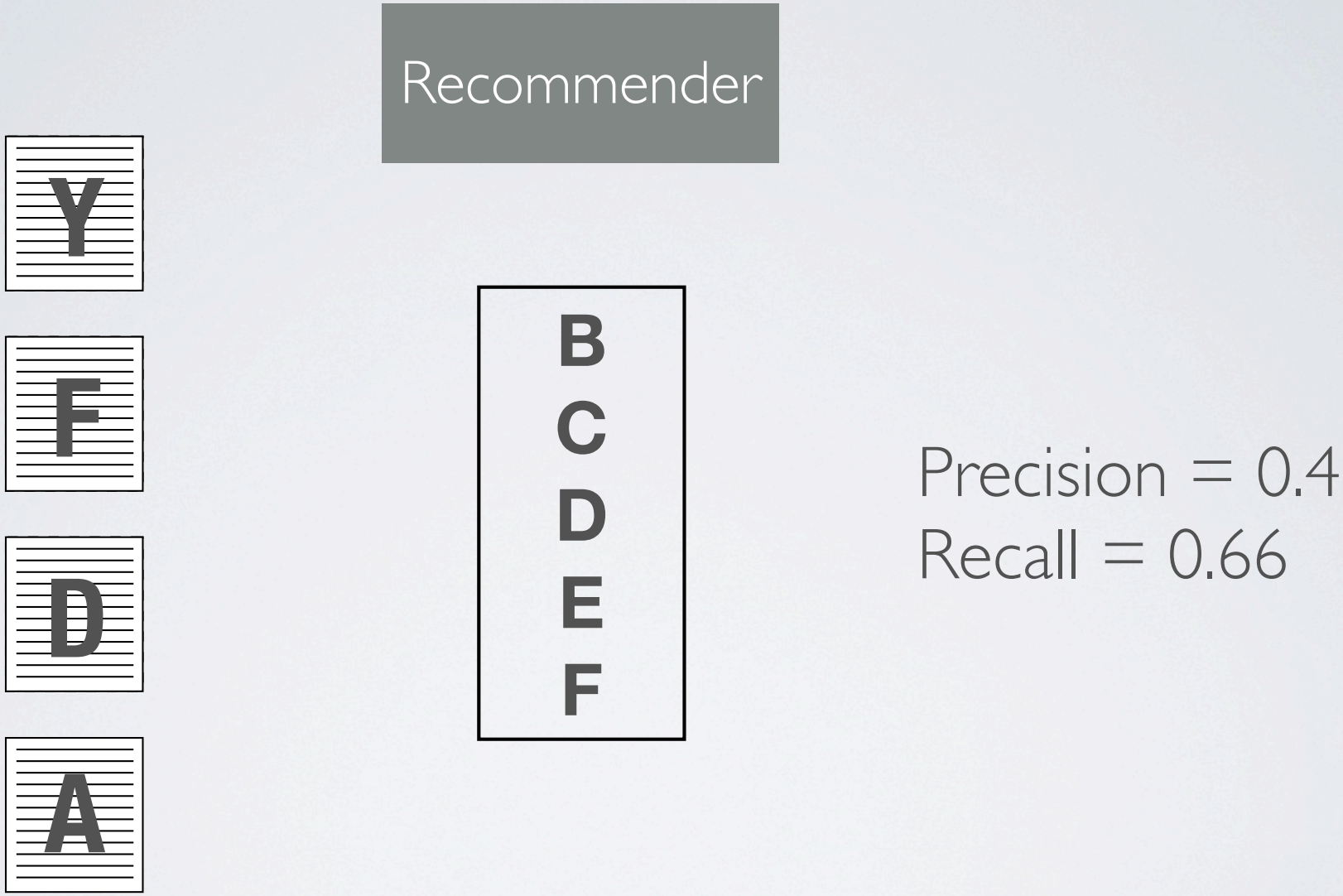


Performance evaluation with precision and recall

Precision	0	0.5	0.25	0.33	0
Recall	0	1	1	1	0



Alternative evaluation scenario–Navigation



Precision = 0.4
Recall = 0.66

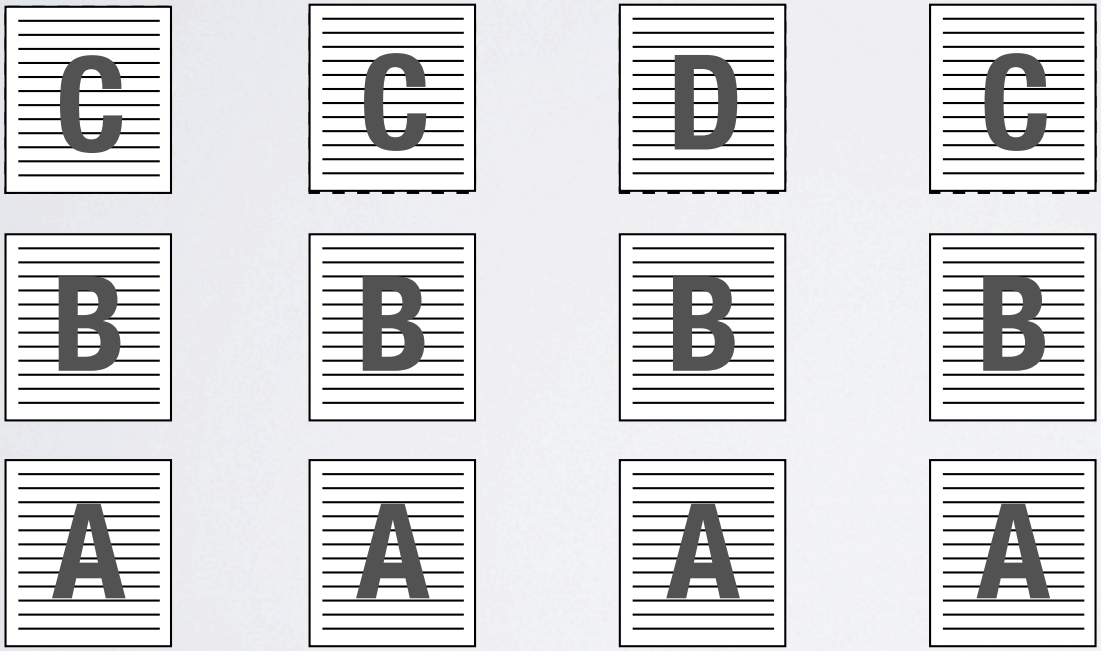
How to choose the entities to predict?

- There is no way to know which entity was changed first from versioning system data.
- The entity to predict (prevention scenario), or the starting entity (navigation scenario), are chosen randomly
- The simplifying assumptions imply that the ordering of selections and queries to a heuristic are immaterial

Hassan and Holt, *Replaying development history to assess the effectiveness of change propagation tools*, EMSE 2006

Association rule mining for change prediction

- Extract association rules from the list of transactions
- Recommend rules with high confidence and support



Confidence = 0.75
Support = 3

Results

	Navigation				Prevention		
Support Count	1				3		
Confidence	0.1				0.9		
Project	Fb	R_M	P_M	L_3	Fb	R_M	P_M
ECLIPSE	0.80	0.36	0.29	0.57	0.04	0.83	0.68
GCC	0.76	0.59	0.35	0.88	0.21	0.97	0.95
GIMP	0.77	0.48	0.28	0.92	0.10	0.94	0.88
JBOSS	0.74	0.36	0.19	0.51	0.03	0.56	0.50
JEDIT	0.95	0.41	0.31	0.88	0.03	0.41	0.37
KOFFICE	0.87	0.45	0.30	0.70	0.04	0.77	0.76
POSTGRES	0.95	0.37	0.29	0.72	0.05	0.72	0.63
PYTHON	0.73	0.46	0.34	0.61	0.04	0.87	0.82
Average	0.82	0.44	0.29	0.72	0.07	0.76	0.70

- Feedback: %of queries with recommendations
- Likelyhood: %of queries with a result in top 3

Zimmermann et al., *Mining Version Histories to Guide Software Changes*, TSE 2005

Types of recommendations

Interestingness	Description
surprising	cross-language
surprising	duplicate code bases
neutral	distant instance creation/being created and call dependence
neutral	distant inheritance
obvious	header-implementation
obvious	interface-implementation
obvious	direct inheritance
obvious	direct call dependence
obvious	same package with less than 20 files

If some recommendation are "obvious", others due, to implicit dependencies or code duplication, are much harder to find otherwise

Ying et al., *Predicting Source Code Changes by Mining Change History,* TSE 2004

Comparison with structural coupling

- Historical Co-change (HIS): entities changing at the same time in the past
- Code Structure (CUD): static dependencies between entities (Call, Use, Define)
- Code Layout (FIL): containment relationships
- Developer Information (DEV): entities changed by the same developer

Project	DEV		HIS		CUD		FIL	
	Recall	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*
NetBSD	0.74	0.01	0.87	0.06	0.37	0.02	0.79	0.16
FreeBSD	0.68	0.02	0.87	0.06	0.40	0.02	0.82	0.11
OpenBSD	0.71	0.02	0.82	0.08	0.38	0.01	0.80	0.14
Postgres	0.78	0.01	0.86	0.05	0.47	0.02	0.77	0.12
GCC	0.79	0.01	0.94	0.03	0.46	0.02	0.96	0.06
All	0.73	0.01	0.87	0.06	0.39	0.02	0.81	0.13
Average	0.74	0.01	0.87	0.06	0.42	0.02	0.83	0.12
F-measure	0.02		0.11		0.04		0.21	

Hassan and Holt, *Replaying development history to assess the effectiveness of change propagation tools*, EMSE 2006

Historical co-change is the best indicator of change propagation

- FREQFIL(80,10): return co-changing entities changing 80% of the time with the initial entities, and entities in the same file, changing at least 70% of the time with the initial entities.

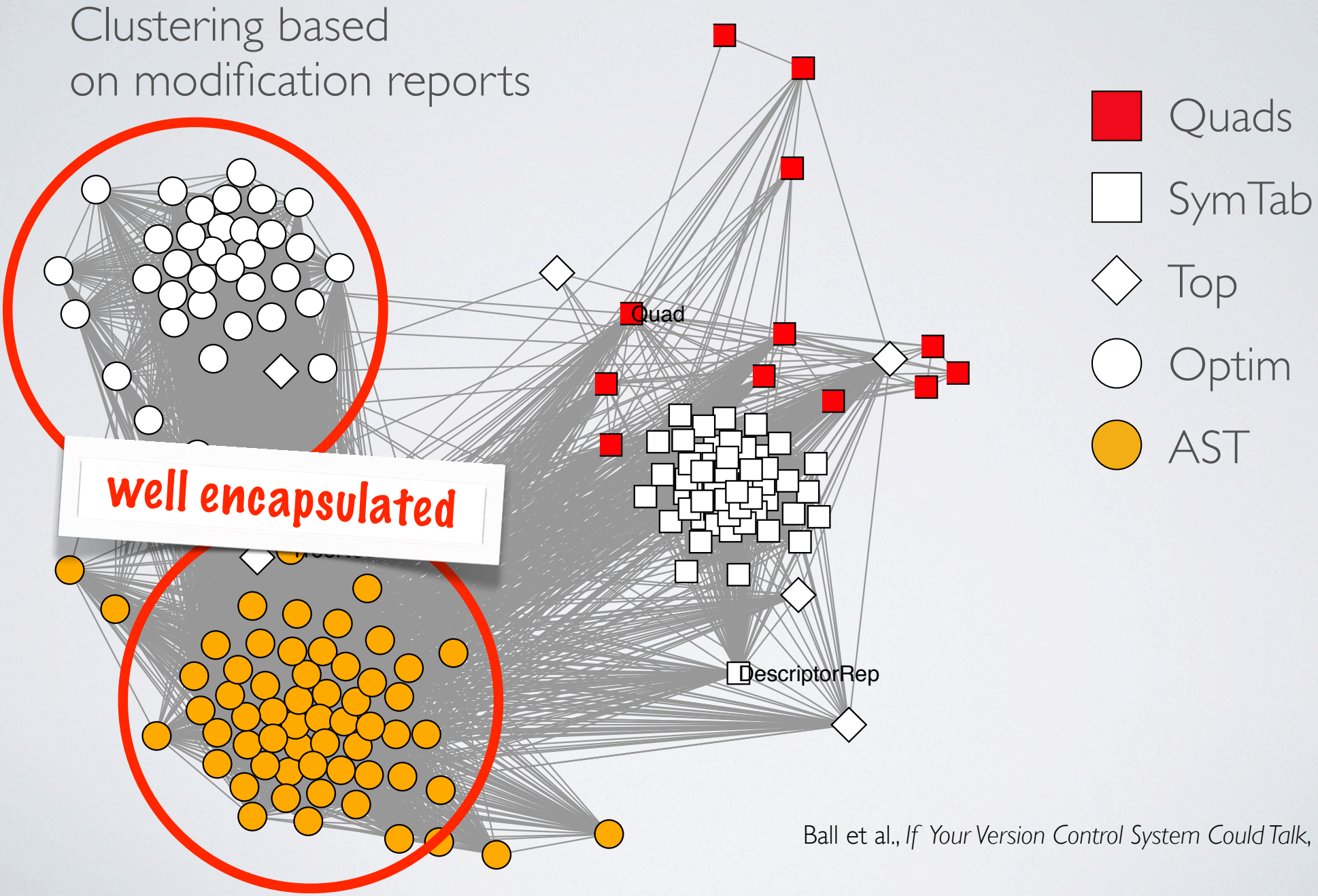
Project	FREQFIL(80, 10)		FREQCUD(70, 10)	
	Recall	*Precision*	*Recall*	*Precision*
NetBSD	0.52	0.51	0.43	0.47
FreeBSD	0.50	0.49	0.40	0.45
OpenBSD	0.50	0.49	0.40	0.46
Postgres	0.48	0.48	0.43	0.44
GCC	0.54	0.48	0.41	0.48
All	0.51	0.49	0.42	0.46
Average	0.51	0.49	0.42	0.46
F-measure	0.50		0.44	

Hassan and Holt, *Replaying development history to assess the effectiveness of change propagation tools*, EMSE 2006

Change (or logical) coupling

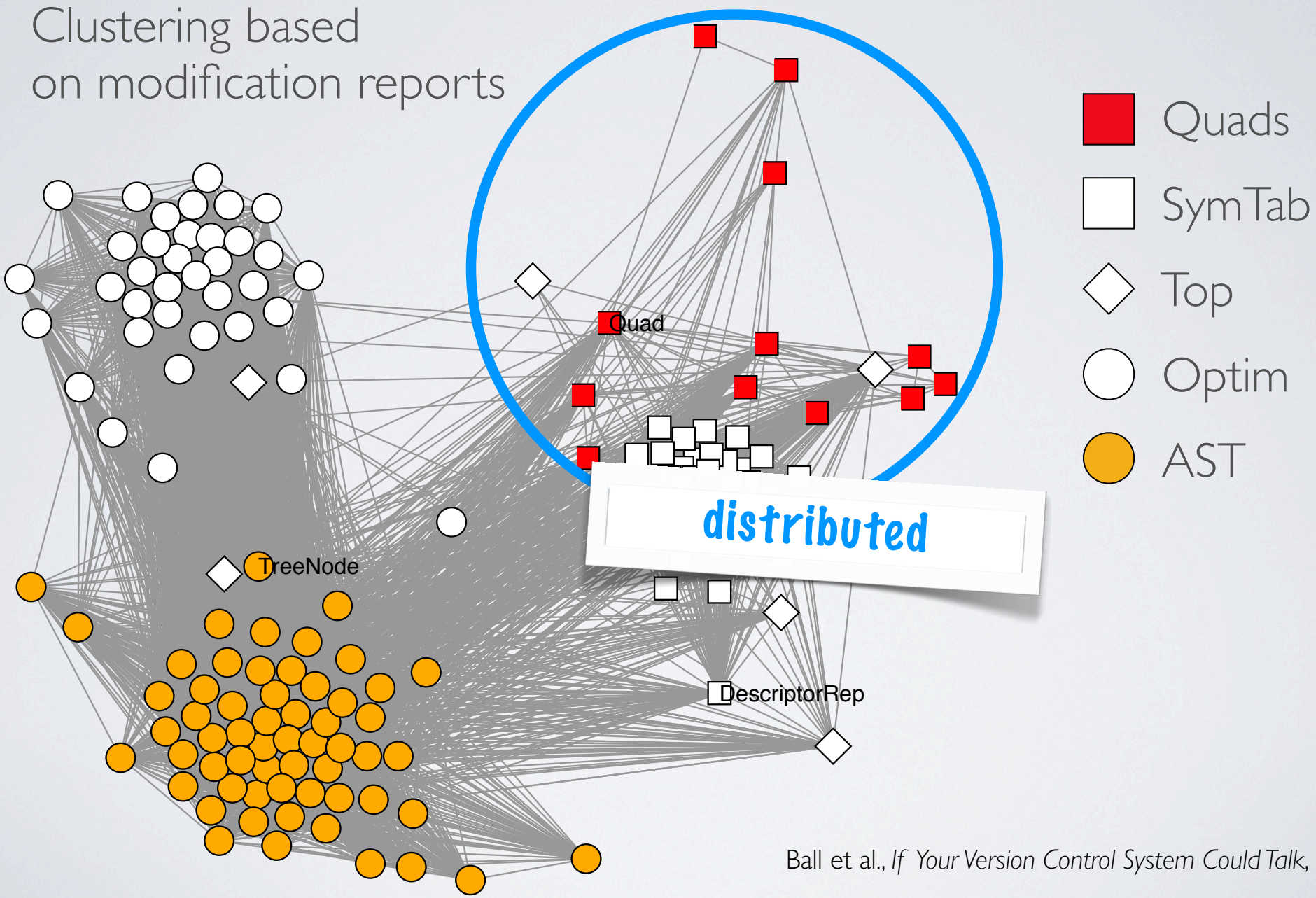
- Two files (classes) are logically coupled if they were frequently modified together
- Two modules are logically coupled if the files they contain are logically coupled
- Logical couplings
  - reveal hidden dependencies not inferable from the source code
  - detect modules that should undergo restructuring
  - identify design problems such as architecture decay
  - pinpoint misplaced artifacts and modularization problems
  - are based on minimal amount of data, rather than MLOC

First work on change coupling



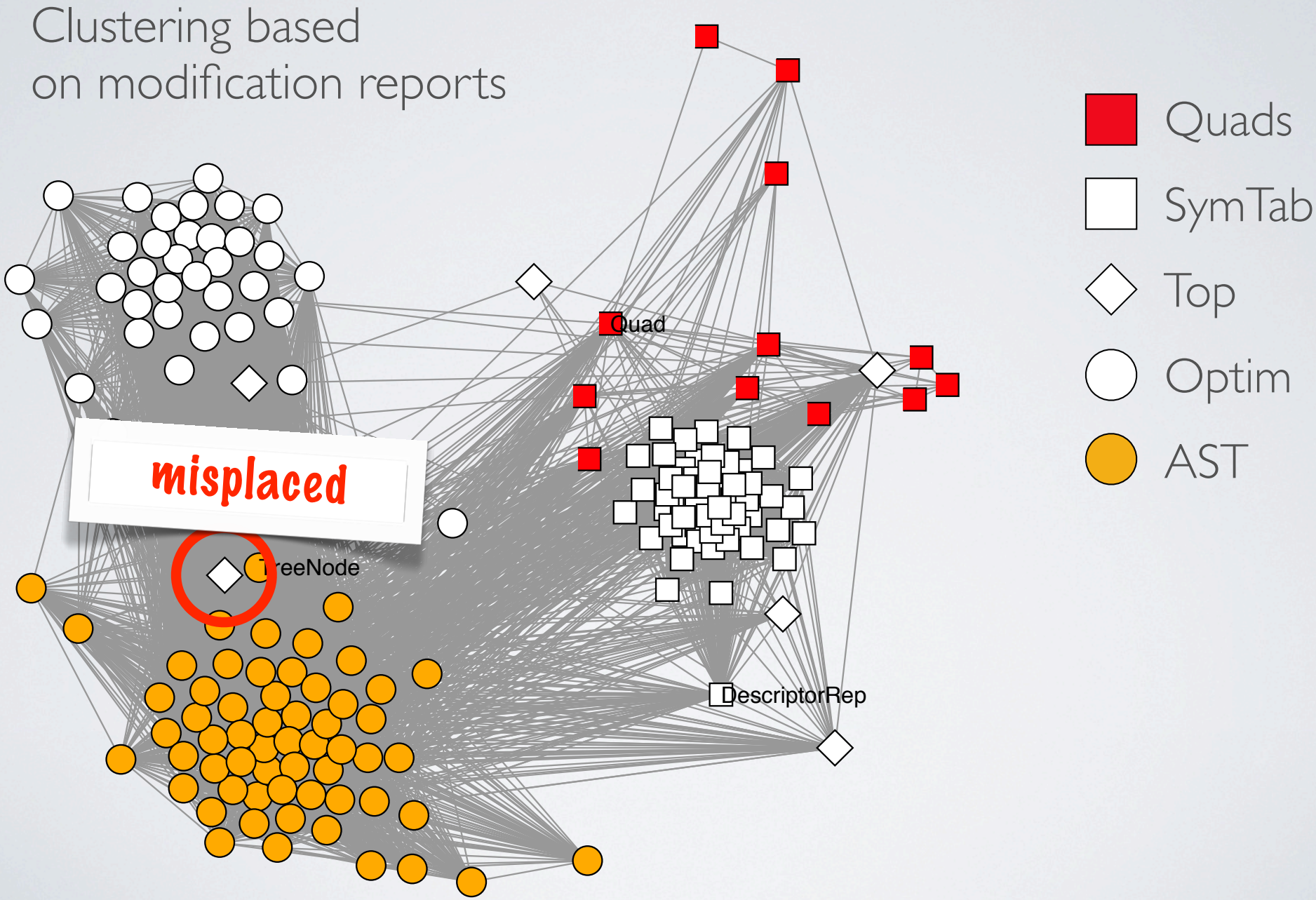
Ball et al., *If Your Version Control System Could Talk*, 1997

First work on change coupling



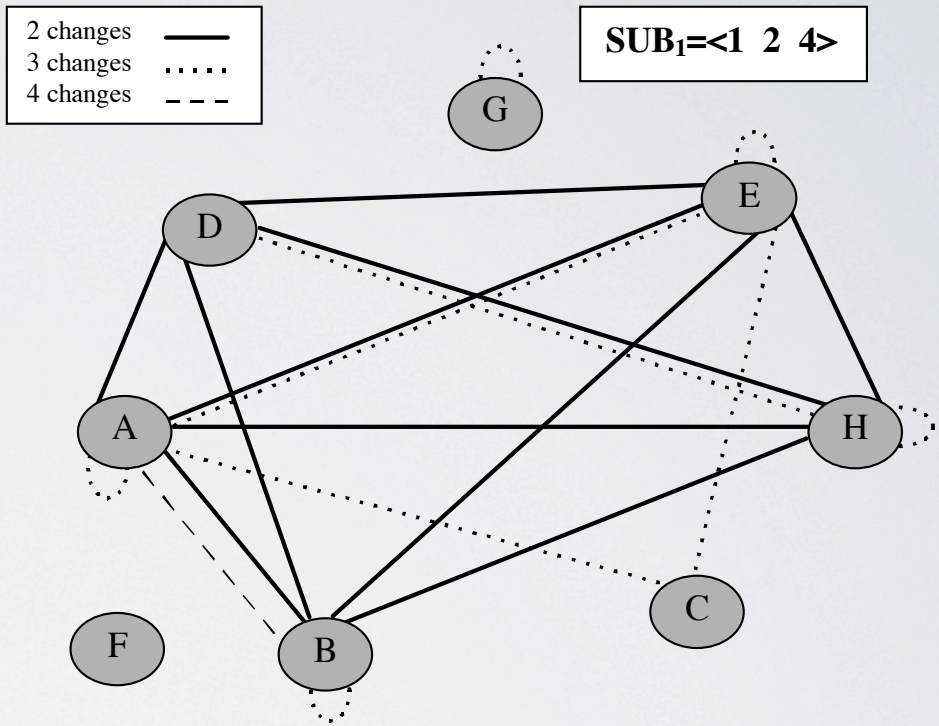
Ball et al., *If Your Version Control System Could Talk*, 1997

First work on change coupling



Definition of logical coupling

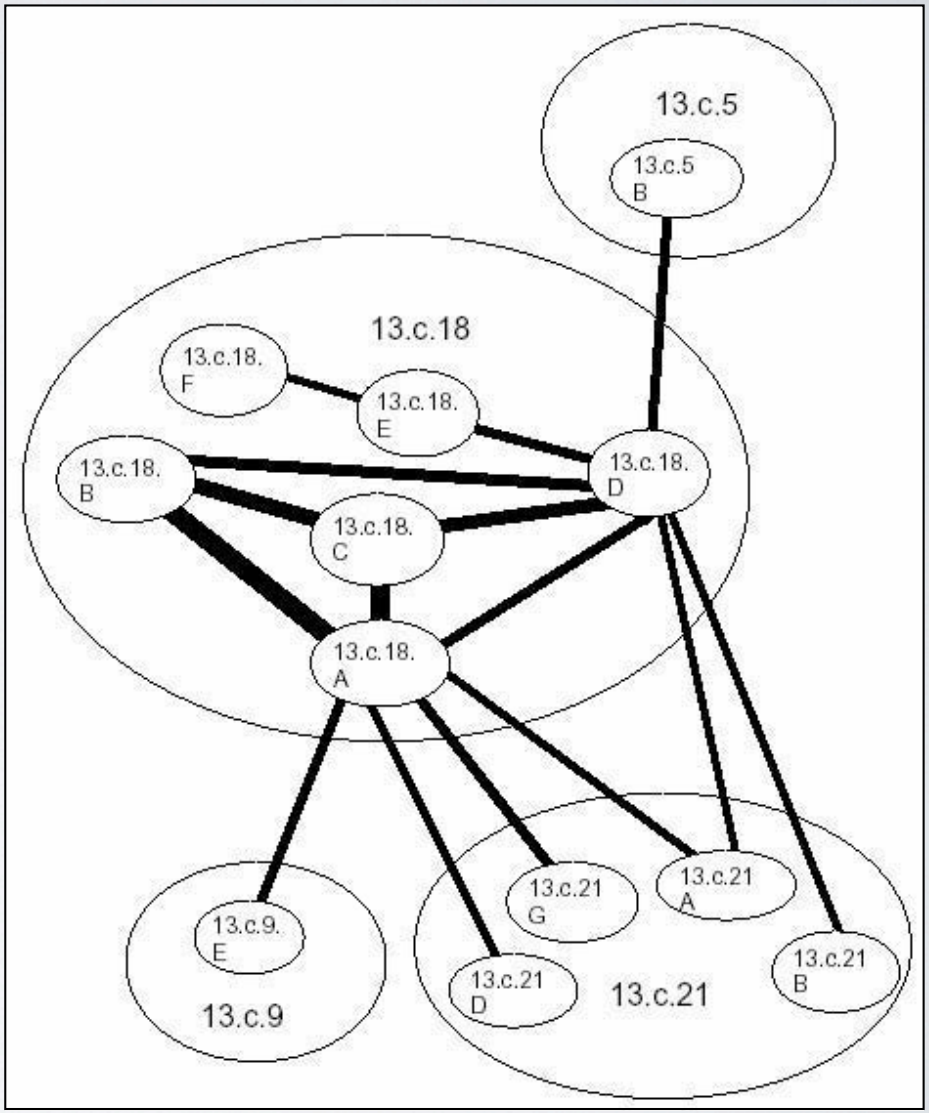
- Gall et al.* identified "logical" couplings among system modules across several releases of a large industrial telecommunication system



Gall et al., *Detection of Logical Coupling Based on Product Release History*, ICSM 1998

Finer grained logical coupling

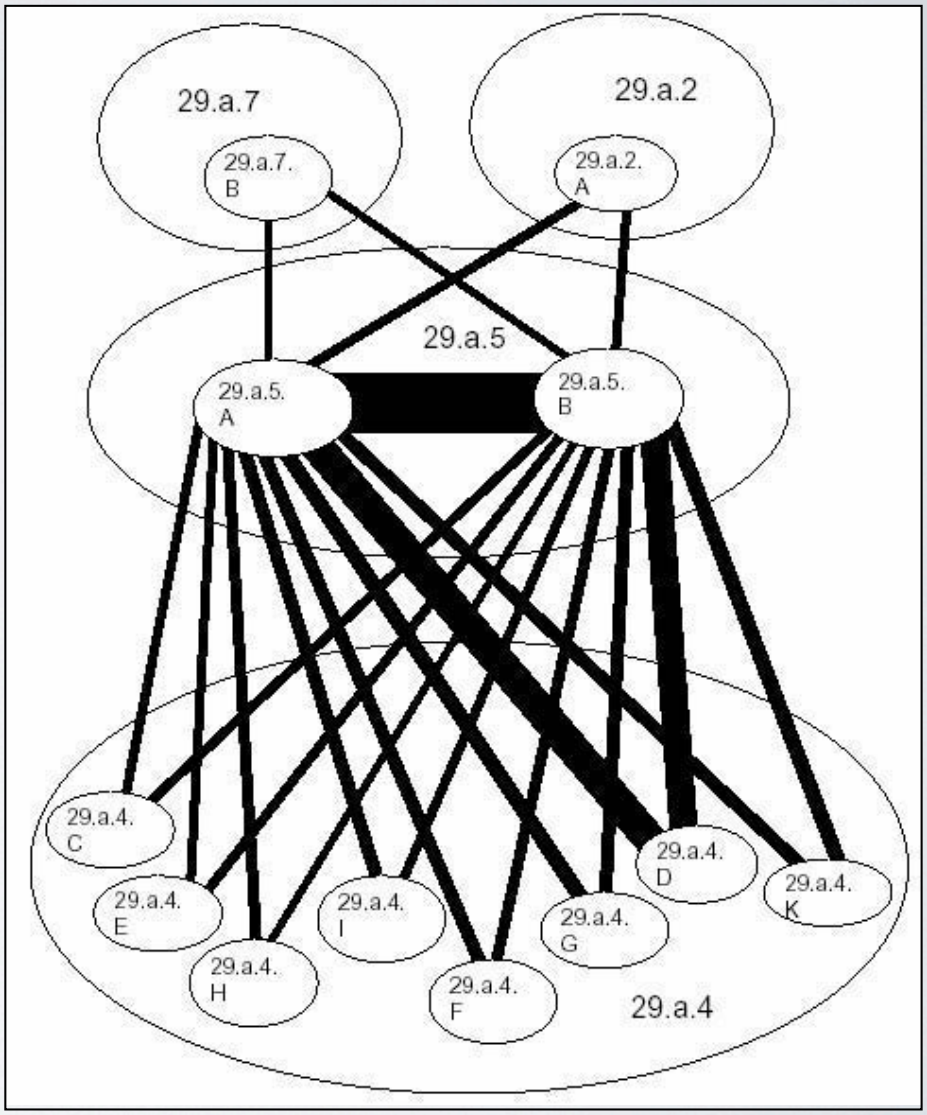
- Couplings among submodules and classes
- To answer questions such as:
  - Which parts of the software system have been changed together most often?
  - How many classes are involved in an external coupling with a particular module?
  - Is the internal or the external coupling stronger?
  - Do some modules have significantly more relations than other modules?



Gall et al., *CVS Release History Data for Detecting Logical Couplings*, IWPSE 2003

Finer grained logical coupling

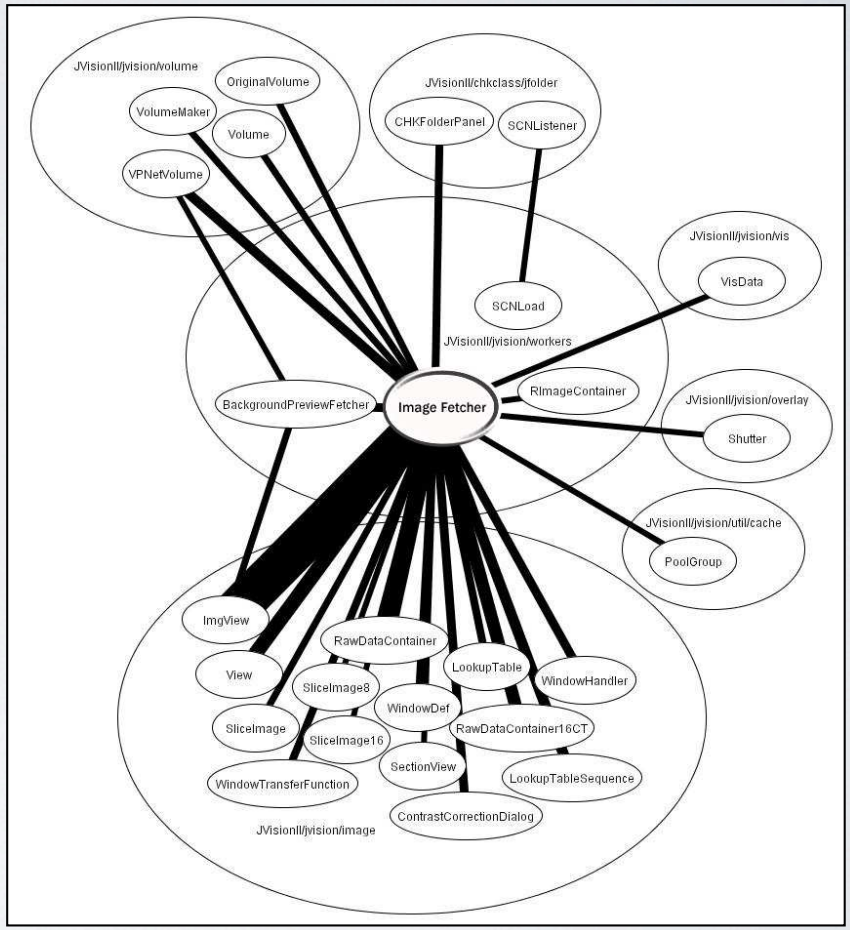
- Couplings among submodules and classes
- To answer questions such as:
  - Which parts of the software system have been changed together most often?
  - How many classes are involved in an external coupling with a particular module?
  - Is the internal or the external coupling stronger?
  - Do some modules have significantly more relations than other modules?



Gall et al., *CVS Release History Data for Detecting Logical Couplings*, IWPSE 2003

Change smells

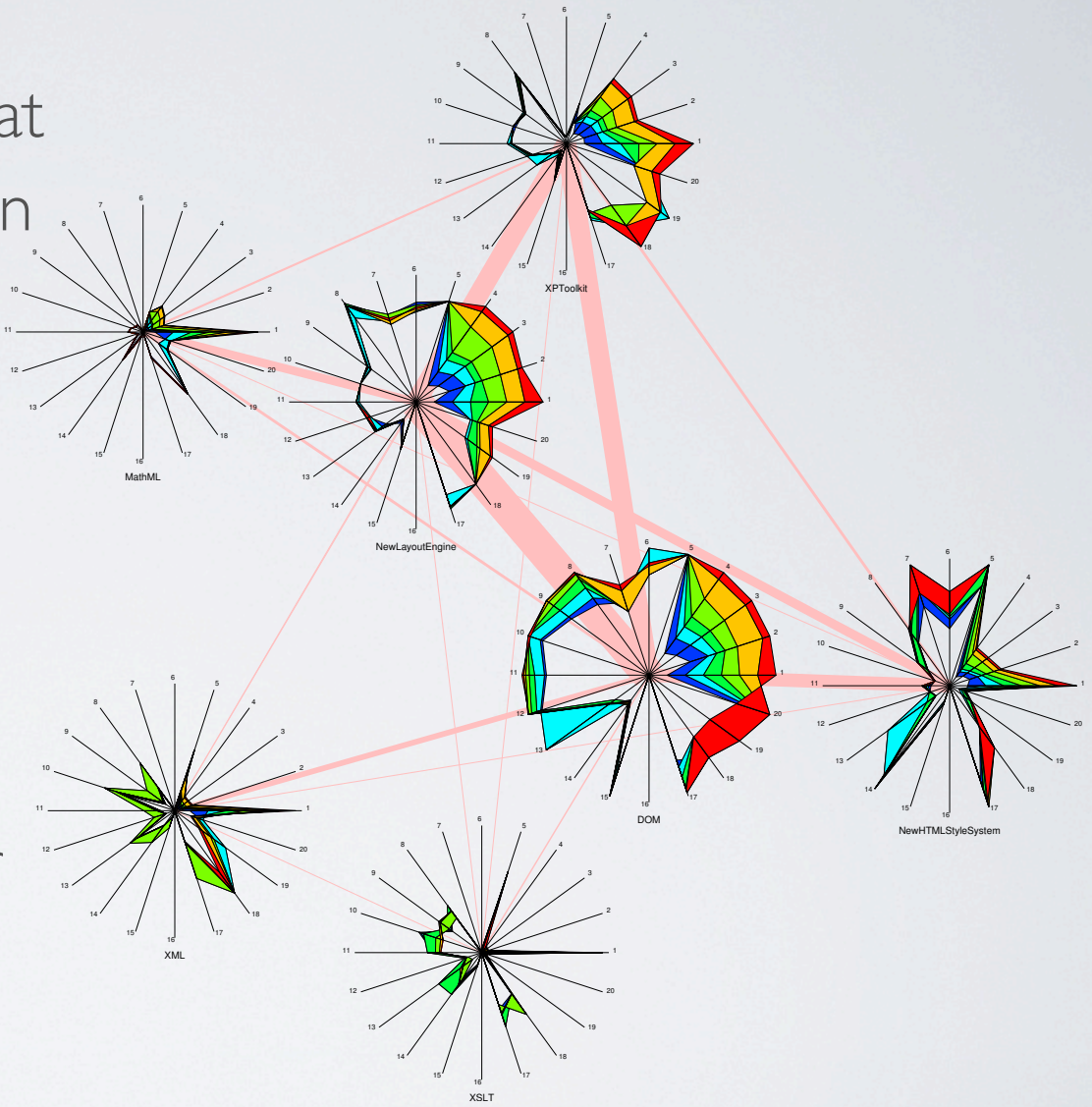
- Change smells are similar to Fowler's bad smells, but based on change coupling data
  - Man in the middle
  - Data container
- Applied refactorings on detected change smells
- Analyzed the evolution of the system after applying the refactorings and showed that the evolvability of the system improved



Ratzinger et al., *Improving evolvability through refactoring*, MSR 2005

Change coupling + metrics

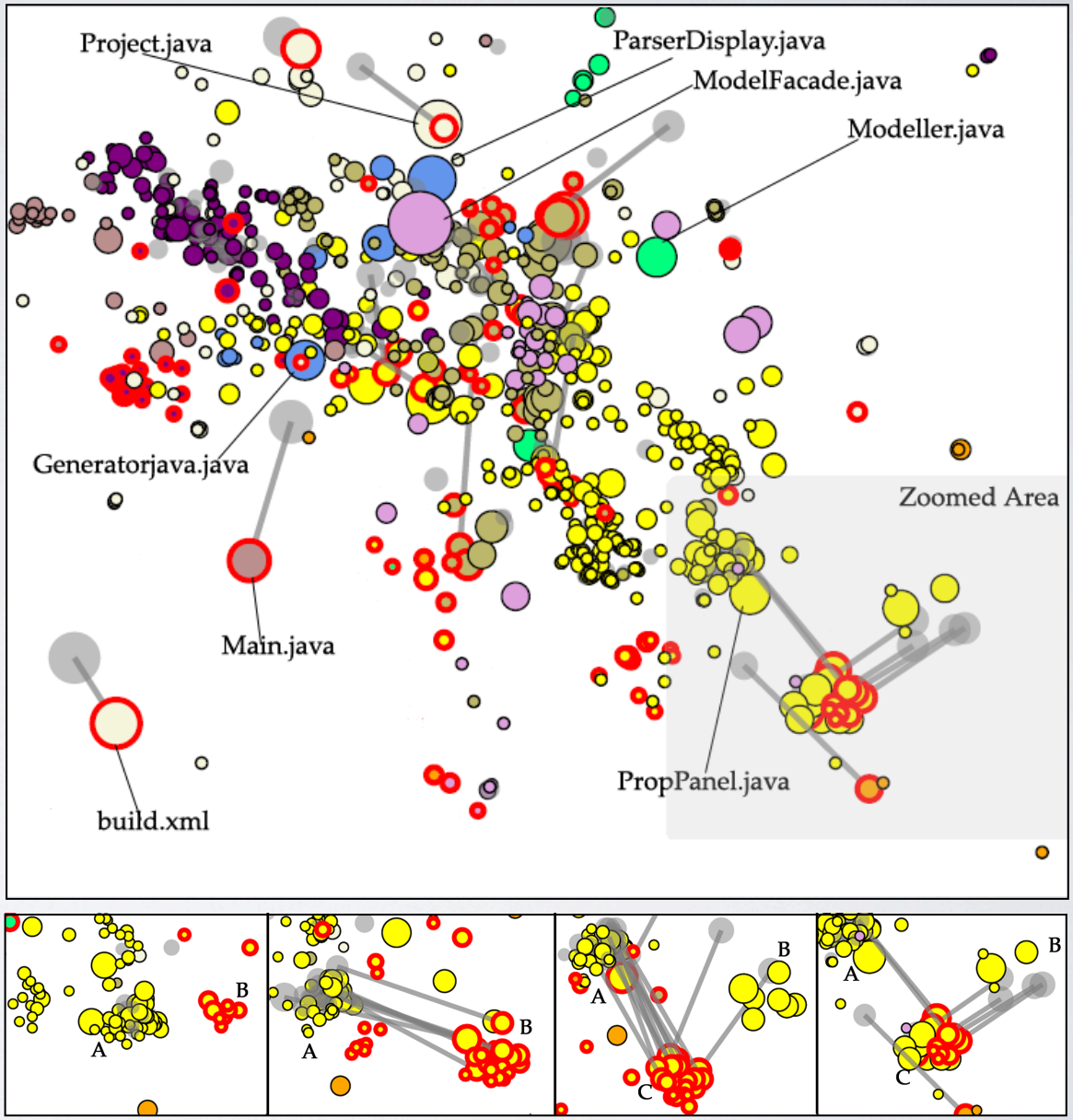
- Modules represented as Kiviat diagrams, mapping metrics on them
- Modules linked by change coupling
- Useful to:
  - Assess maintenance cost of modules
  - Study the stability of modules



Pinzger et al., *Visualizing multiple evolution metrics*, SoftVis 2005

Animated clustering of change coupling data

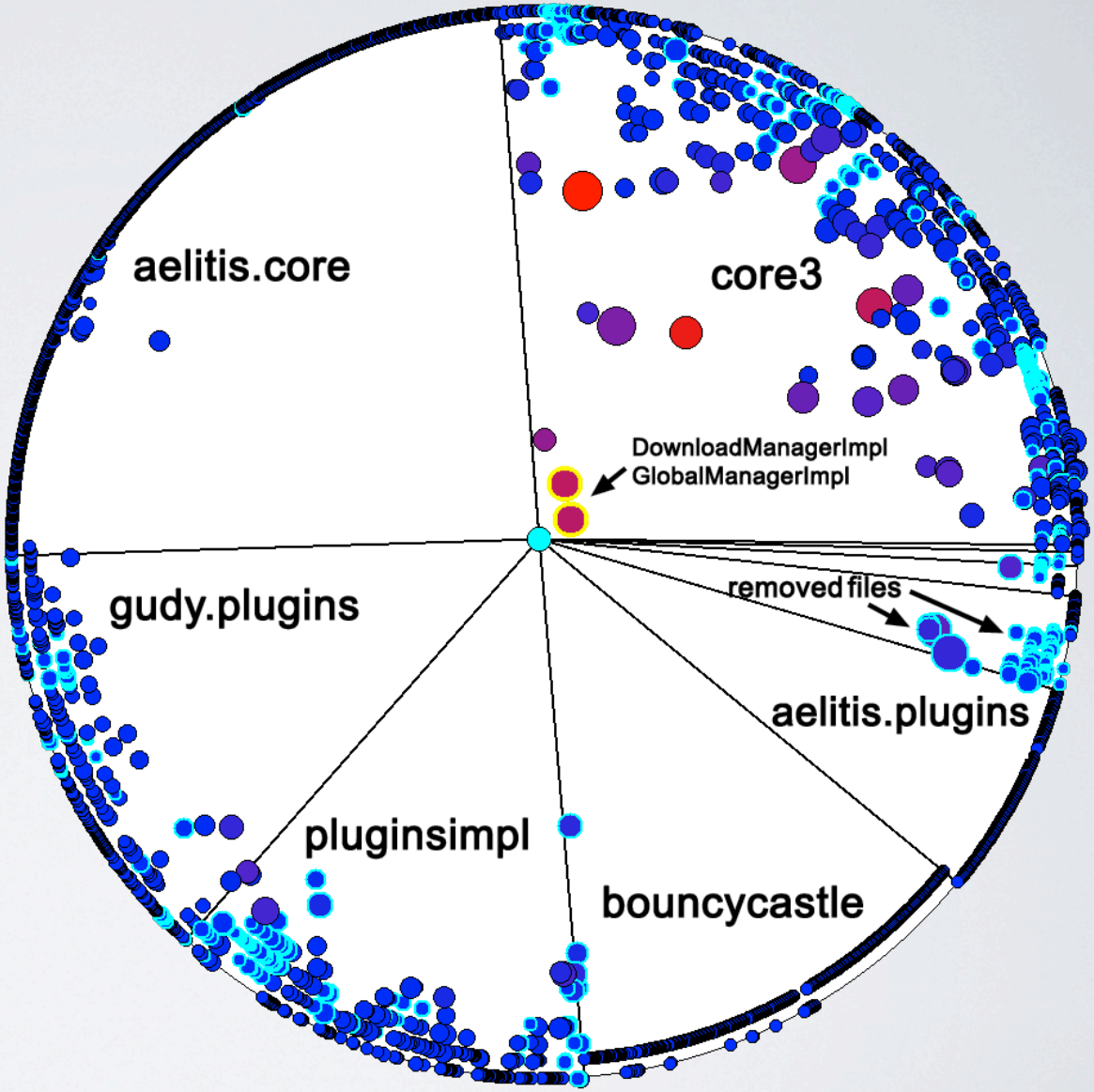
- Animated clustering is useful for:
  - understanding decay symptoms (since when did they exist?)
  - highlighting refactoring candidates (cross-cutting artifacts)
  - spotting good structure (artifacts that always change together)



Beyer and Hassan, *Animated visualization of software history using evolution storyboards*, WCRE 2006

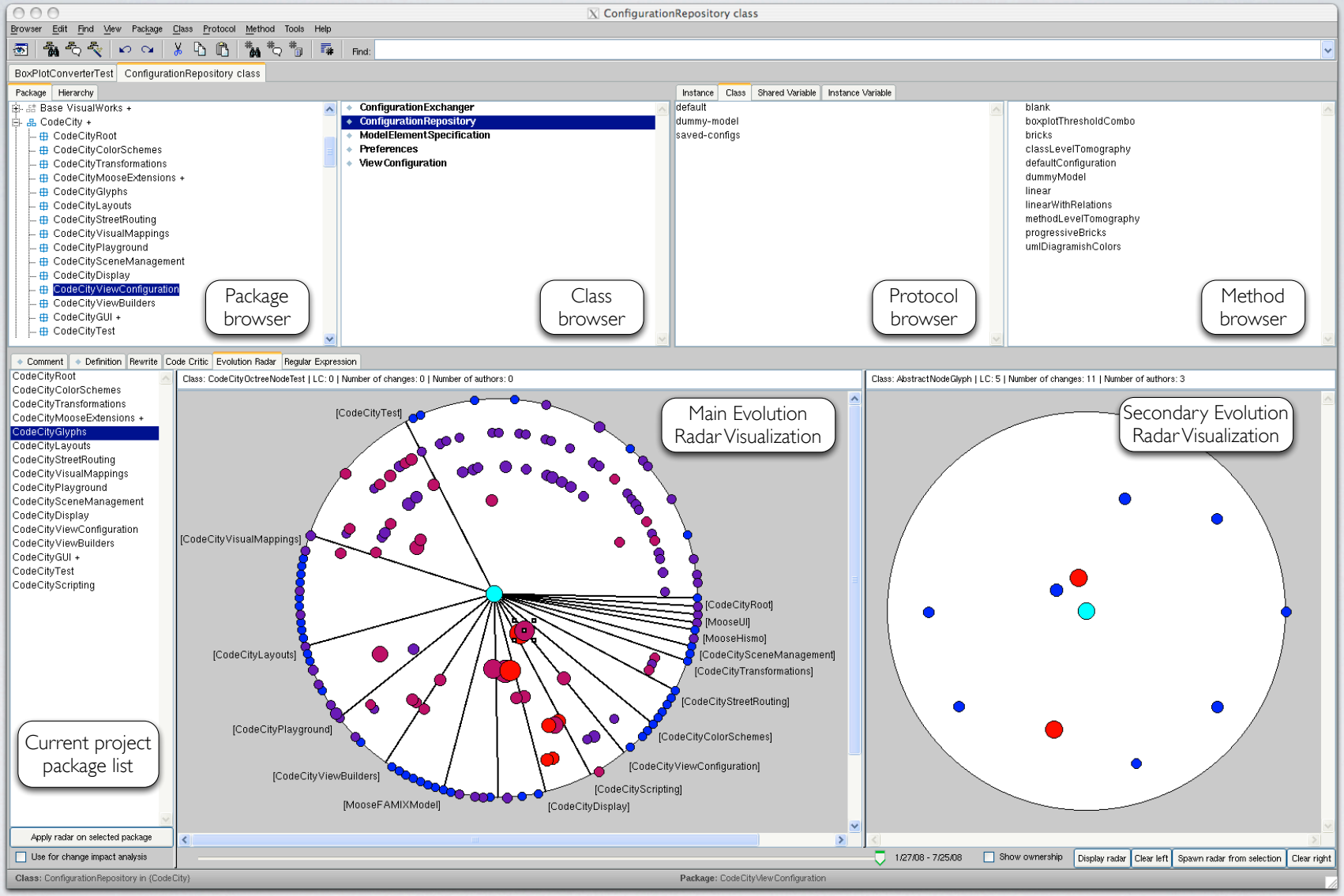
Integrated change coupling data

- File- and module-level change coupling
- The evolution radar supports:
  - assessing architecture decay
  - change impact analysis
  - redocumentation and restructuring



D'Ambros et al., *Visualizing Co-Change Information with the Evolution Radar,* TSE 2009

Integrated change coupling IDE integration



- redocumentation and restructuring

D'Ambros et al., *Visualizing Co-Change Information with the Evolution Radar*, TSE 2009

Improving detection of change coupling

- Several heuristics are employed
  - Time window size
  - Weighting changes in the past
  - How many artifacts are changed together (e.g. filter out license update)
  - How much is “frequently”?

Improvement with DTW

- The authors defined the "Synchrony" pattern for files that change together similarly in the time dimension
- To detect the synchrony pattern they used DTW
  - Dynamic time warping (DTW) measures the similarity between two sequences which may vary in time or speed ("warping" the sequences non-linearly in the time dimension)
  - In our context the sequences are series of changes to software artifacts

Bouktif et al., *Extracting change-patterns from CVS repositories*, WCRE 2006

Improvement with DTW

- DTW provides an improvement in precision and recall with respect to previous approaches (e.g., Zimmermann et al. TSE 2005)
- External precision (evaluated by system experts) 78.98%, external recall 76.89%.
- Internal precision on Mozilla 73.58%, recall 67.61%

Bouktif et al., *Extracting change-patterns from CVS repositories*, WCRE 2006

Improvement with macro co-change

- Macro co-changes (MCCs): artifacts that co-change with large intervals
- Dephase macro co-changes (DMCCs): MCC that always happen with the same shifts in time
- Detection of (D)MCCs
  - for each artifact a bit vector is created, where each bit represents a time interval
    - 1 → the artifact was changed during the corresponding time interval
    - 0 → no modifications
  - (D)MCC are identified based on the Hamming distance among bit vectors
- Evaluation on 4 long-lived open source systems
  - quantitative, comparing the approach with association rules in terms of precision and recall
  - qualitatively, by using bug tracking systems and mailing lists to confirm couplings

Jaafar et al., *A Novel Analysis of Co-change*, WCRE 2011

Change analysis and prediction

- Goal: Analyze software repository data to:
  - support change propagation and predict future changes
  - detect design problems (e.g. architecture decay)


Defect prediction
Change analysis and prediction
Empirical studies
Expertise & bug triaging
Visual evolution analysis
Human factors

Empirical studies

- Goal: Analyze software repositories data to:
  - empirically validate/invalidate common wisdom
  - investigate a particular phenomenon
  - provide actionable suggestions based on empirical evidence

Growth of OSS

- Previous studies* indicated that the growth of software systems tend to decrease as they get larger
- Godfrey and Tu empirically investigated this assumption on the Linux Kernel**

*Gall et al., *Software evolution observations based on product release history*, ICSM 1997

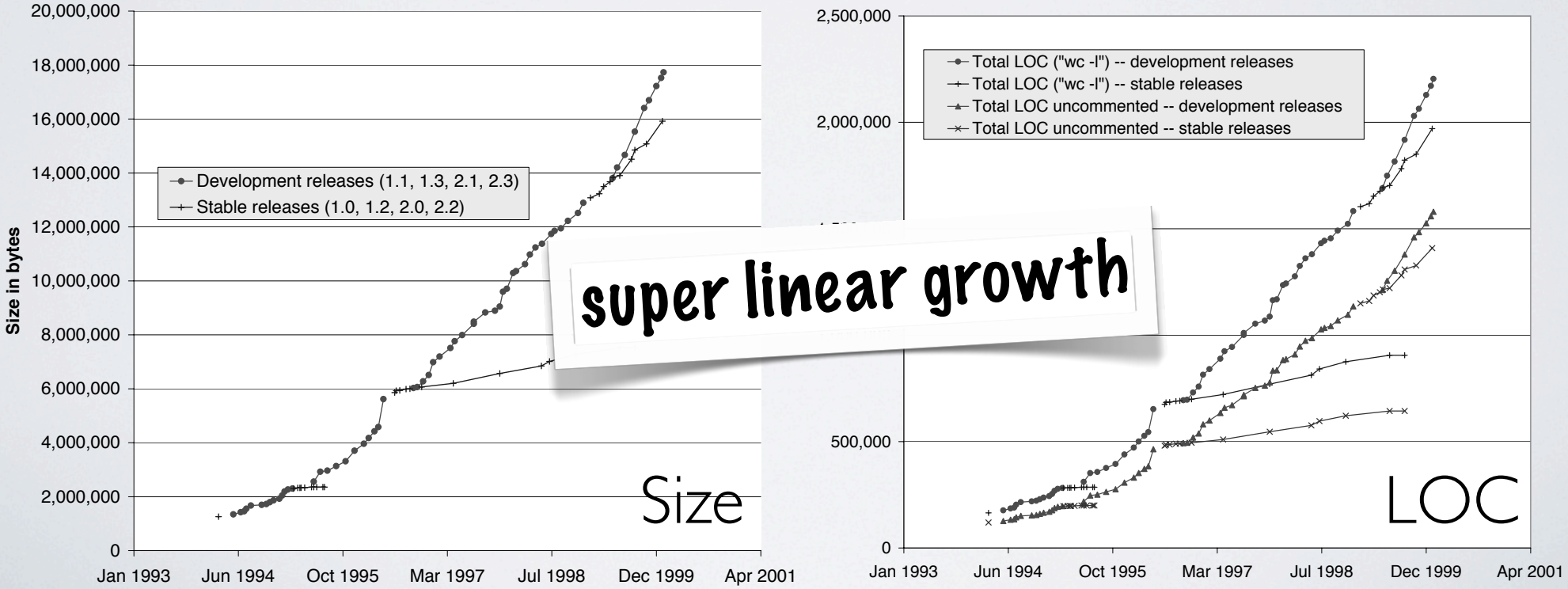
*Lehman et al., *Implications of evolution metrics on software maintenance*, ICSM 1998

*Turski, *Reference model for smooth growth of soft- ware systems*, TSE 1996

**Godfrey and Tu, *Evolution in open source software: A case study*, ICSM 2000

Growth of OSS

- Previous studies* indicated that the growth of software systems tend to decrease as they get larger
- Godfrey and Tu empirically investigated this assumption on the Linux Kernel**



How can such a large system grow so fast?

- The answer is in the data
  - >50% of code is device drivers, which are independent
  - The same features are implemented multiple times for different CPU/architectures
  - The core of the Linux kernel is much smaller than the code base

Open source (OSS) vs commercial software

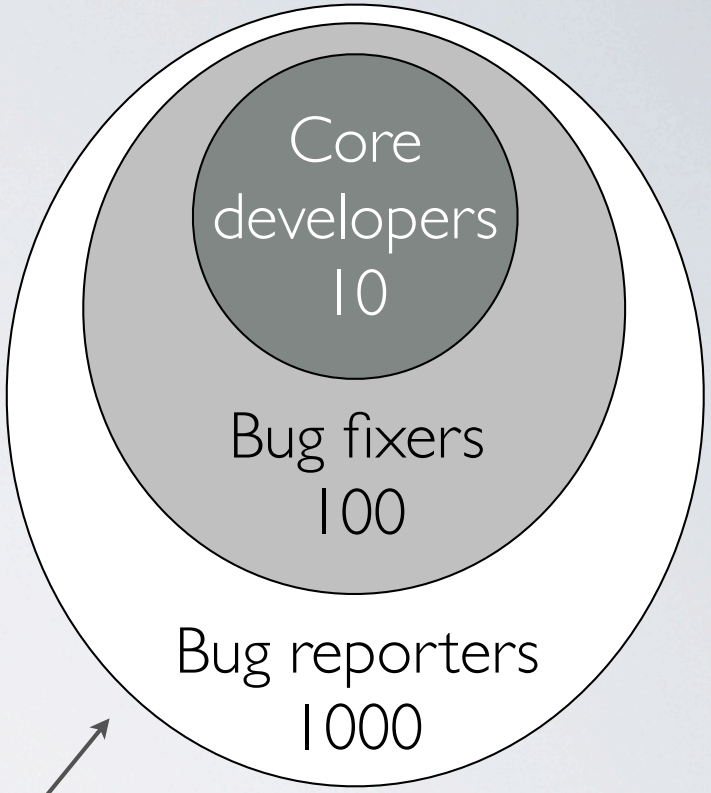
- Common wisdom: OSS can compete successfully with traditional commercial development methodologies
- Mockus et al. empirically verified this wisdom on Apache
- They collected developer mailing lists, code change history and problem reports
  - Differently from other MSR approaches, all the data was extracted from email archives (emails were automatically generated for each commit and for each bug report)

Most influential paper award

Mockus et al., *A case study of open source software development: the Apache server*, ICSE 2000

Findings: OSS development process

- OSS have a core of 10-15 developers which produces 80% of the code
  - For projects so large that 15 people cannot handle the code base, satellite projects are created (e.g. Apache-SSL)



- Projects that do not follow this model are likely to fail, because there are not enough people to find/fix bugs

Findings: Detecting and fixing bugs

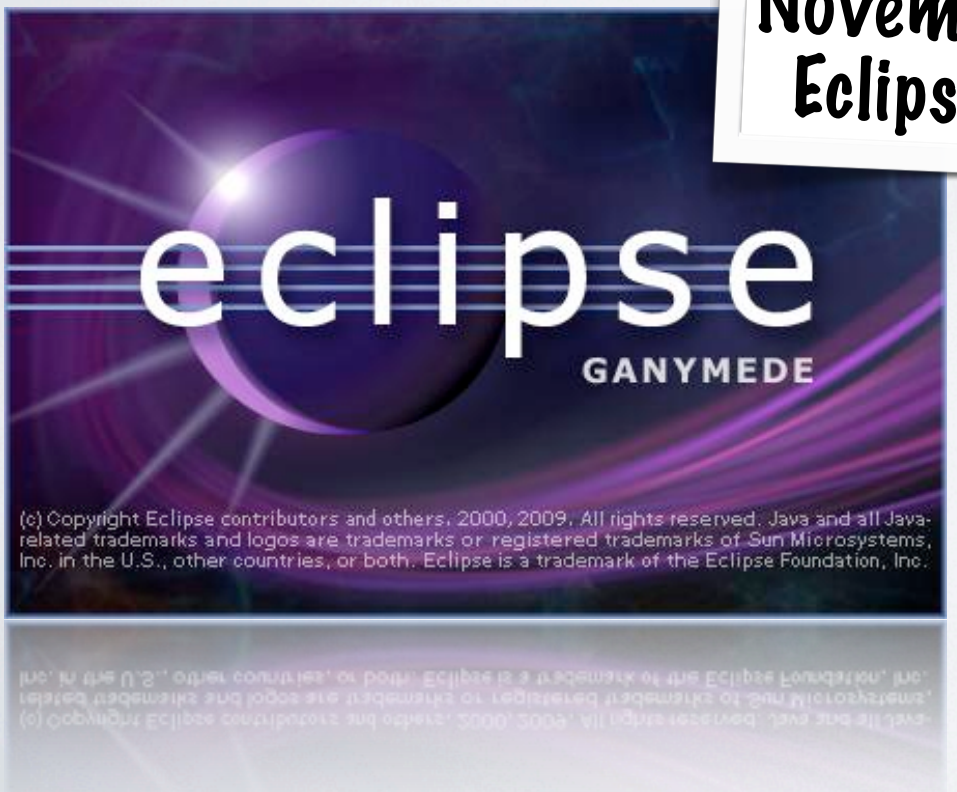
- In OSS defect density is lower than in commercial software (given the same level of testing)
- Developers are also users
  - Avoiding the lack of domain knowledge
- OSS has a faster responses to users' problems
  - Patches can be distributed as soon as they are developed (not bundled into new releases / service packs)

Conclusion

- Commercial and OSS practices might be fruitfully hybridized

June 2000

November 2001
Eclipse as OSS



Clones and software quality

- Clones are considered bad smells which indicate poor maintainability*:
  - clones can silently replicate hidden bugs**
  - cloning is often performed hastily without a full understanding of the cloned code
- Rahman et al. empirically assess whether clones decrease software quality by making the code more defect prone+

*Fowler et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley 1999

**Jiang et al., *Context-based detection of clone-related bugs*, ESEC/FSE 2007

+Rahman et al., *Clones: What is that Smell?*, MSR 2010

Clones and software quality

- Clones are considered bad smells which indicate poor maintainability*:
  - clones can silently replicate hidden bugs**
  - cloning is often performed hastily without a full understanding of the cloned code
- Rahman et al. empirically assess whether clones decrease software quality by making the code more defect prone[+]


Gimp


Nautilus


Evolution

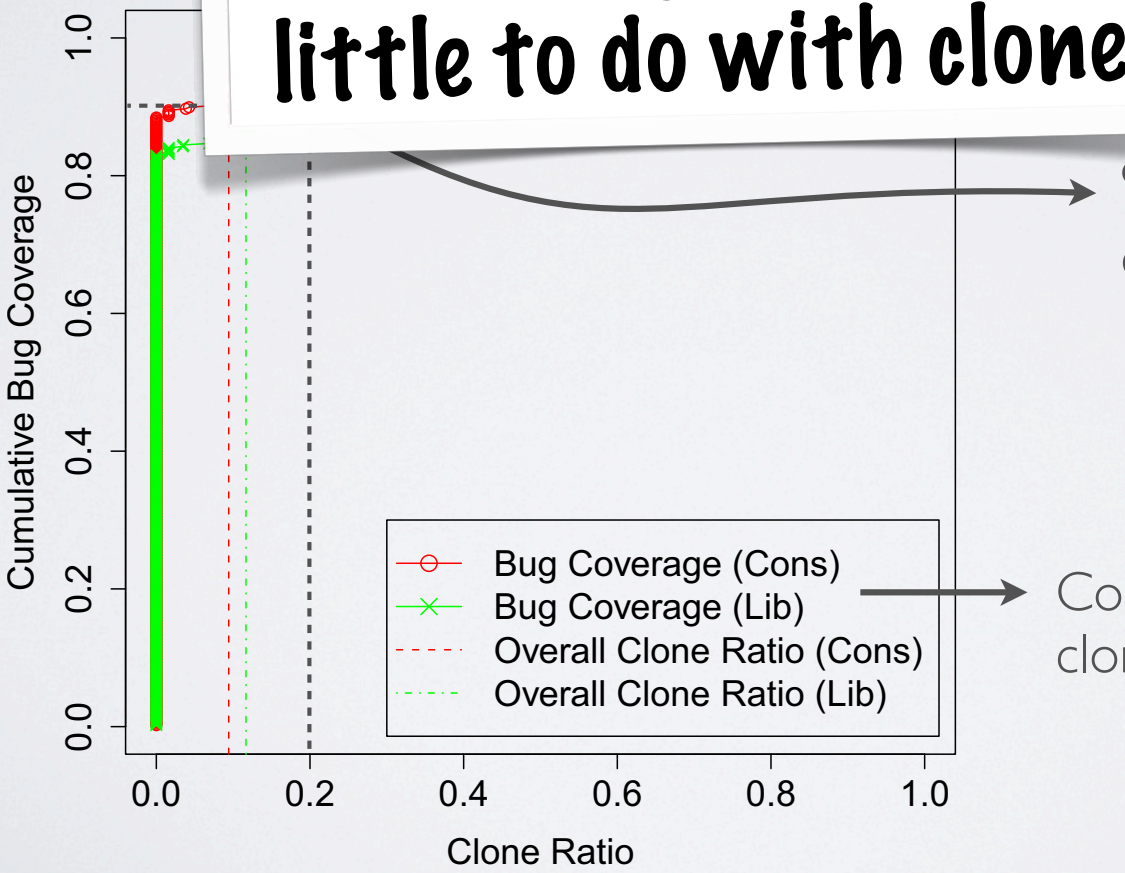

HTTPD

[+]Rahman et al., *Clones: What is that Smell?*, MSR 2010

Does cloned code contribute to bugs?

$$\text{bug clone ratio} = \frac{\text{buggy code part of clones}}{\text{buggy code}}$$

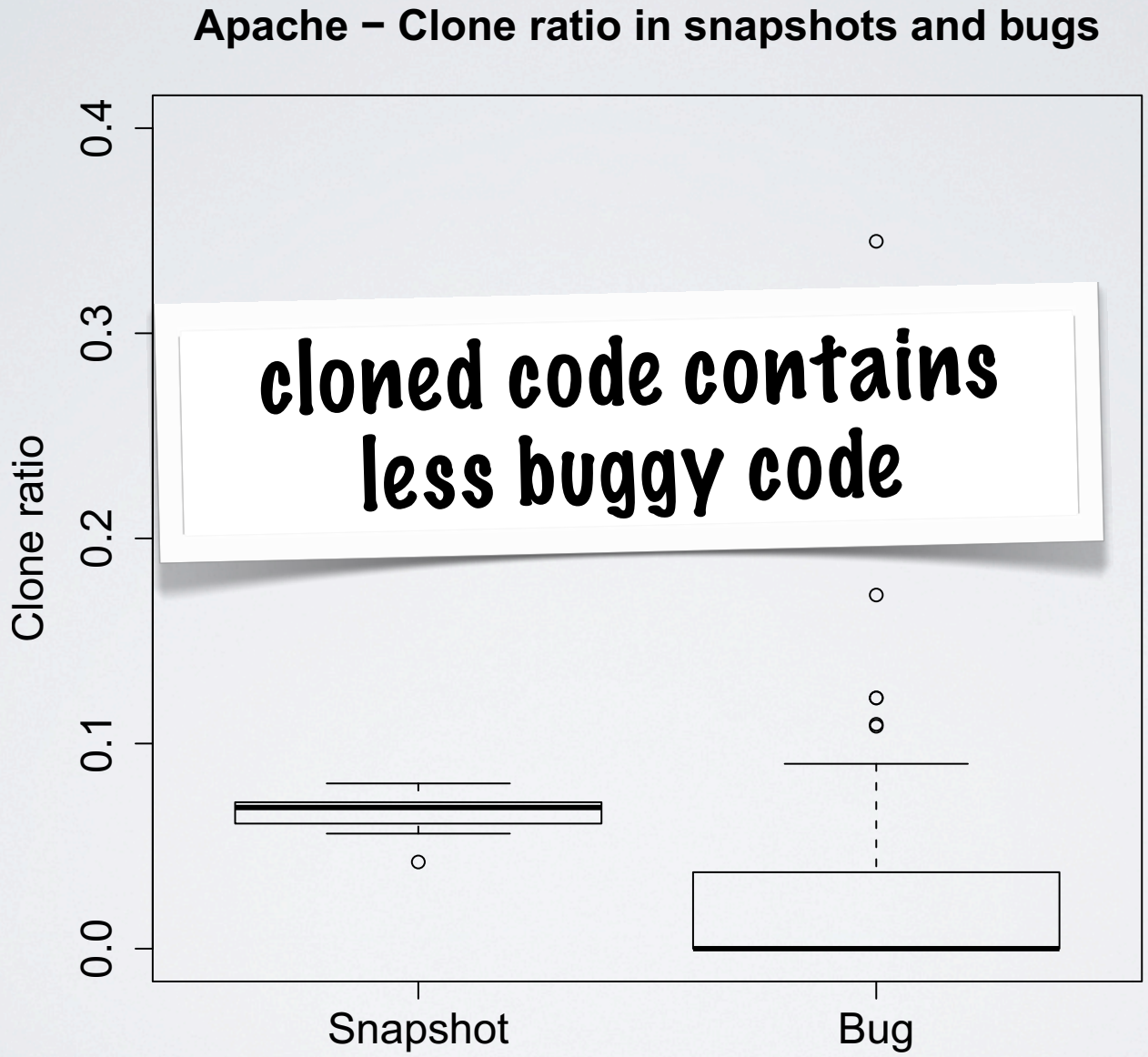
most bugs have very little to do with clones



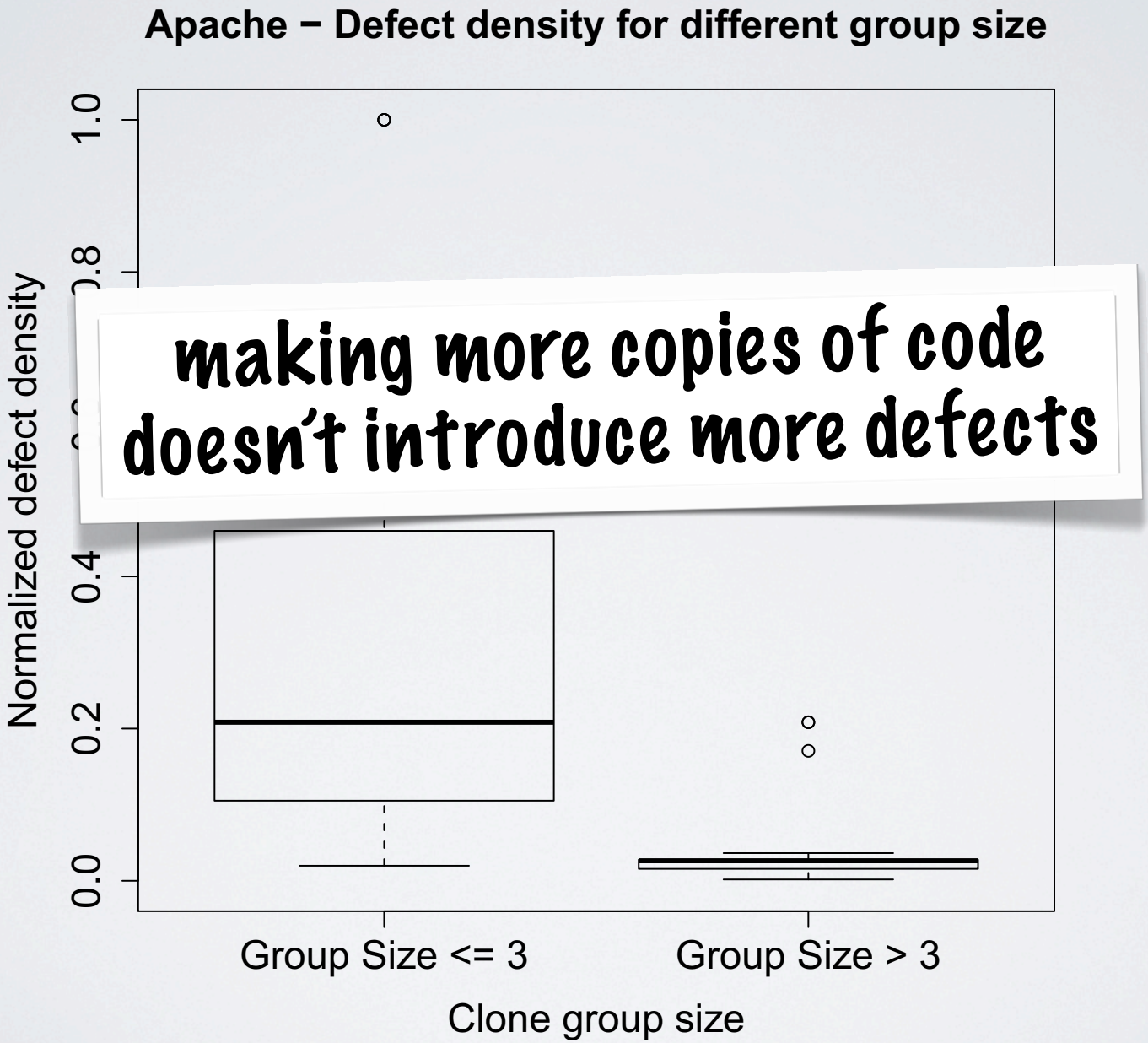
90% of bugs have 20% or less bug clone ratio

Conservative and liberal clone detection

Do clones occur more often in buggy code?



Are larger clone groups more buggy?



making more copies of code doesn't introduce more defects

Effects of distributed development on software quality*

- Distributed development is more challenging and riskier than collocated development for several reasons*: delayed feedback, restricted communication, difficulty of synchronous communication, inconsistent build environments, lack of trust and confidence between sites, etc.
- Bird et al. studied the effects of distributed development on software quality, measured in terms of of post-release failures, on Windows Vista**
  - Distributed binaries: when at least 25% of the commits came from locations other than where binary's owner resides
  - Collocated binaries: produced by collocated teams (the rest)
  - The distribution of post-release failures is studied in the two populations

**Bird et al., *Does distributed development affect software quality? an empirical case study of windows vista*, ICSE 2009

*Olson and Olson, *Distance matters*, Human- Computer Interaction 2000

Results

- There is a statistically significant (Mann-Whitney test) difference between the two populations for the average number of post-release failures per binary
- Since the difference was small (~8%), a further investigation was performed
  - By controlling for team size, it was shown that the difference becomes negligible
  - Hypothesis: binaries that are selected for distributed development are smaller and/or less complex
    - No significant difference was found between distributed and collocated binaries in terms of several properties (e.g., code metrics, code churn, test coverage)

Possible explanations

- Sites were not in competition
- Cultural barriers were alleviated because a number of engineers from the US site visited the Indian one for an extended period of time
- Lack of communication was mitigated by the fact that Vista developers made heavy use of synchronous communication daily (employees stayed at work late or arrived early for a status conference call)
- The same production tools (configuration management and build system) are used throughout all Microsoft sites
- Distributed ownership was minimized since Microsoft enforces strong code ownership
- Common release dates and milestones
- The organizational structure spans geographical locations at low levels

Effects of code ownership on software quality

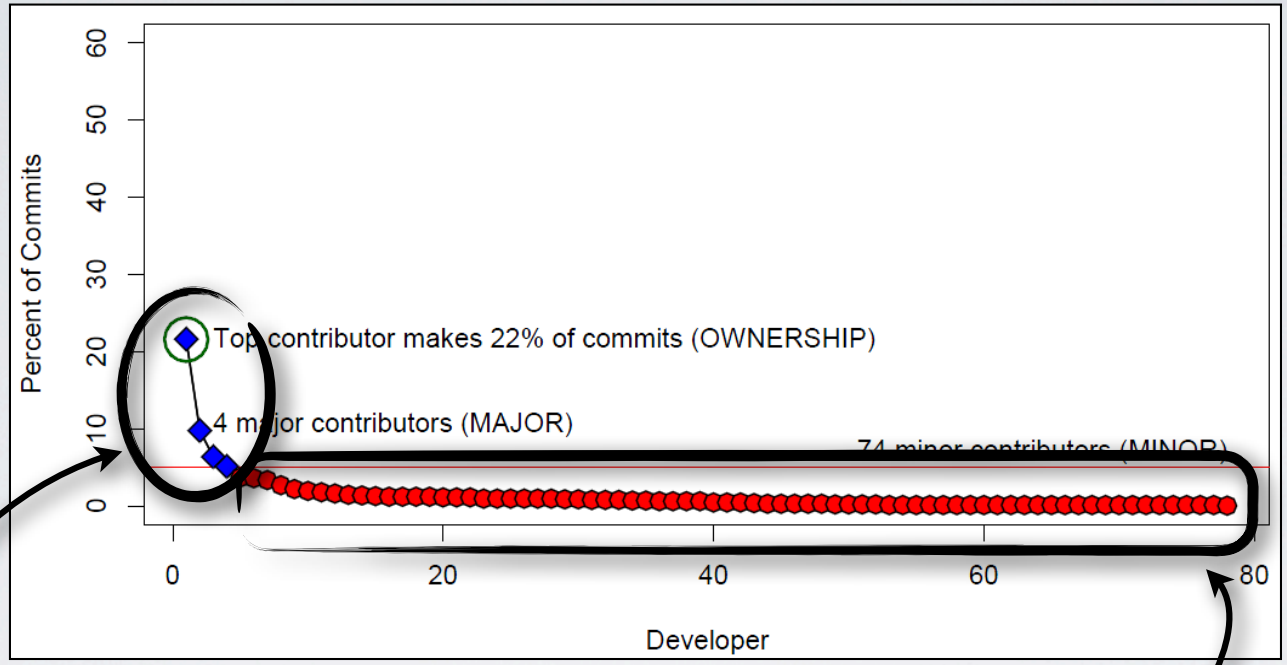
- The number of people working on a software artifact is correlated with the number of bugs affecting it*
- Why and how?
- Bird et al. empirically analyzed the relationship between various ownership measures and bugs in Windows Vista and Windows 7**
- For a software artifact they defined
  - major contributor: >5% of commits
  - minor contributor: <5% of commits

*Bird et al., *Does distributed development affect software quality? an empirical case study of windows vista*, ICSE 2009

*Nagappan et al., *The influence of organizational structure on software quality: an empirical case study,* ICSE 2008

**Bird et al., *Don't Touch My Code! Examining the Effects of Ownership on Software Quality*, ESEC/FSE 2011

Effects of code ownership on software quality



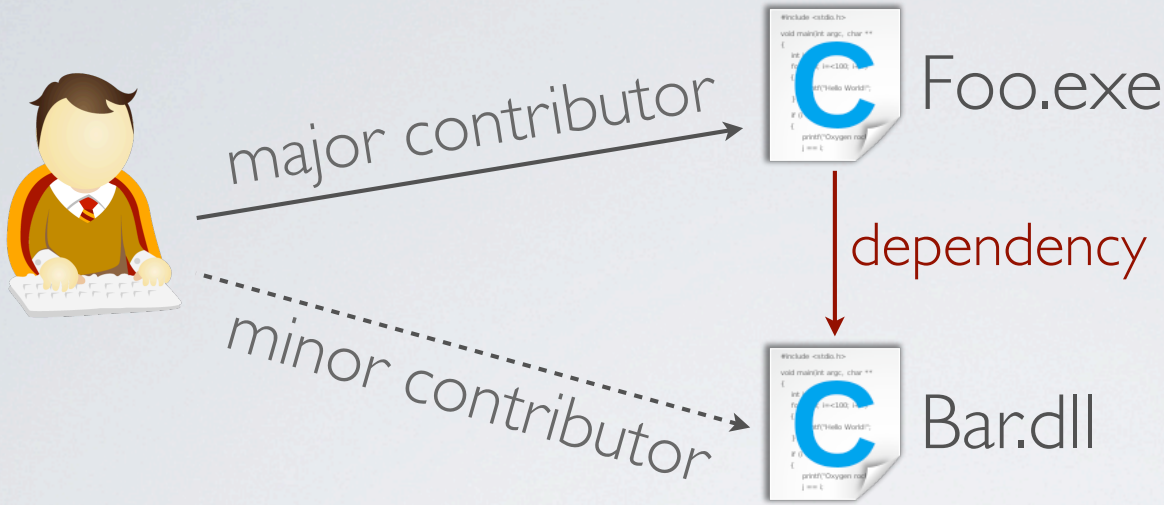
- For a software artifact they defined
  - major contributor: >5% of commits
  - minor contributor: <5% of commits

**Bird et al., *Don't Touch My Code! Examining the Effects of Ownership on Software Quality*, ESEC/FSE 2011

The level of ownership affects software quality

- Software artifacts with many minor contributors are more defect-prone
  - Also if controlling for well known quality factors such as size and complexity
- Software artifacts with high ownership are less defect-prone

Minor contributors are the key



- Minor contributors to an artifact are often major contributors to another artifact
- The contributions of minor contributors are likely to introduce defects
- Actionable conclusions:
  - Changes performed by minor contributors should be carefully reviewed
  - Minor contributors should ask the desired changes to be implemented by major contributors

Effects of change coupling on software quality

- Change coupling was considered harmful as it leads to architecture decay
- D'Ambros et al. empirically investigated the impact of change coupling on a tangible effect of low software quality, i.e., the presence of software defects
  - is change coupling correlated with software defects?
  - can we use change coupling data for defect prediction?

D'Ambros et al., *On the Relationship Between Change Coupling and Software Defects*, WCRE 2009

Measuring change coupling

Distribution

Number of coupled classes

Force

Number of co-changes

Time decay

Changes far in the past count less

Is change coupling correlated with defects?

Distribution

Force

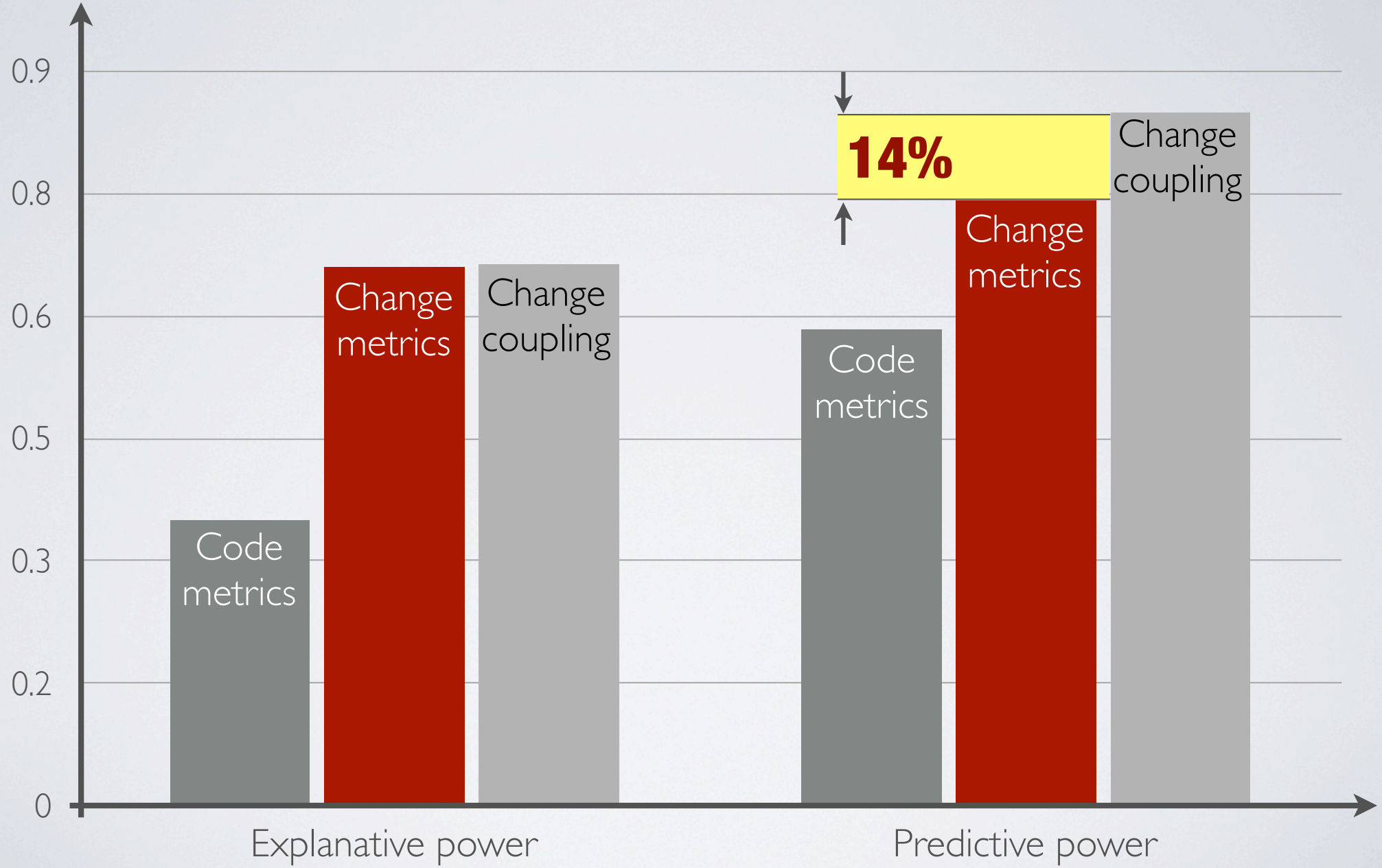
Change coupling strongly correlates with software defects

~~Time~~ decay

0.8+

Spearman's correlation

Can we use change coupling for defect prediction?

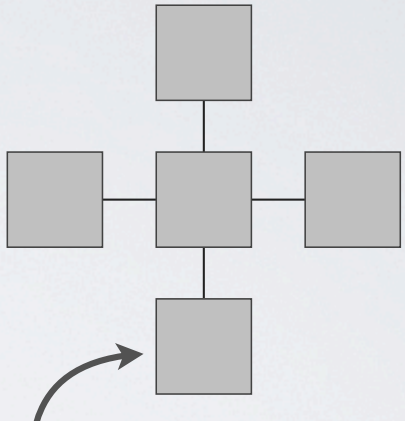


Effects of design flaws on software quality

- Common wisdom: design flaws are a symptom of low software quality
- D'Ambros et al. empirically studied the relationship between software defects and
  - the presence of a catalog of design flaws
  - the addition of a catalog of design flaws over a system's evolution

D'Ambros et al., *On the Impact of Design Flaws on Software Defects*, QSIC 2010

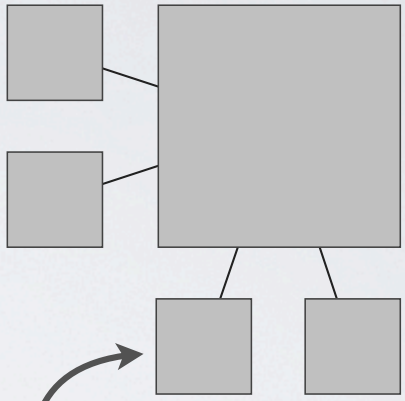
Design flaws



Class (Size ∝ responsibility)

Design guideline: A class should have one responsibility

Design flaws



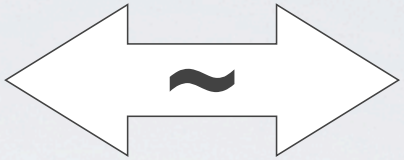
Class (Size ∝ responsibility)

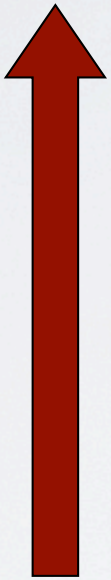
Design guideline: A class should have one responsibility

Violated

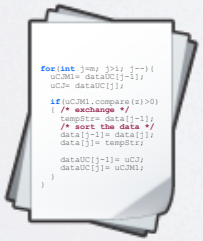
Analyzing design flaws

Design flaws



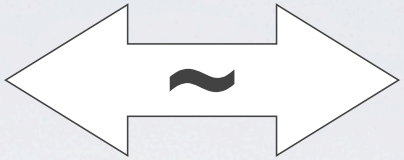


Source code



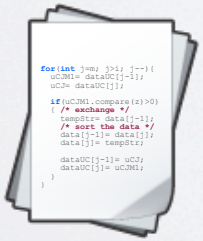
Analyzing design flaws

Design flaws



Detection strategies

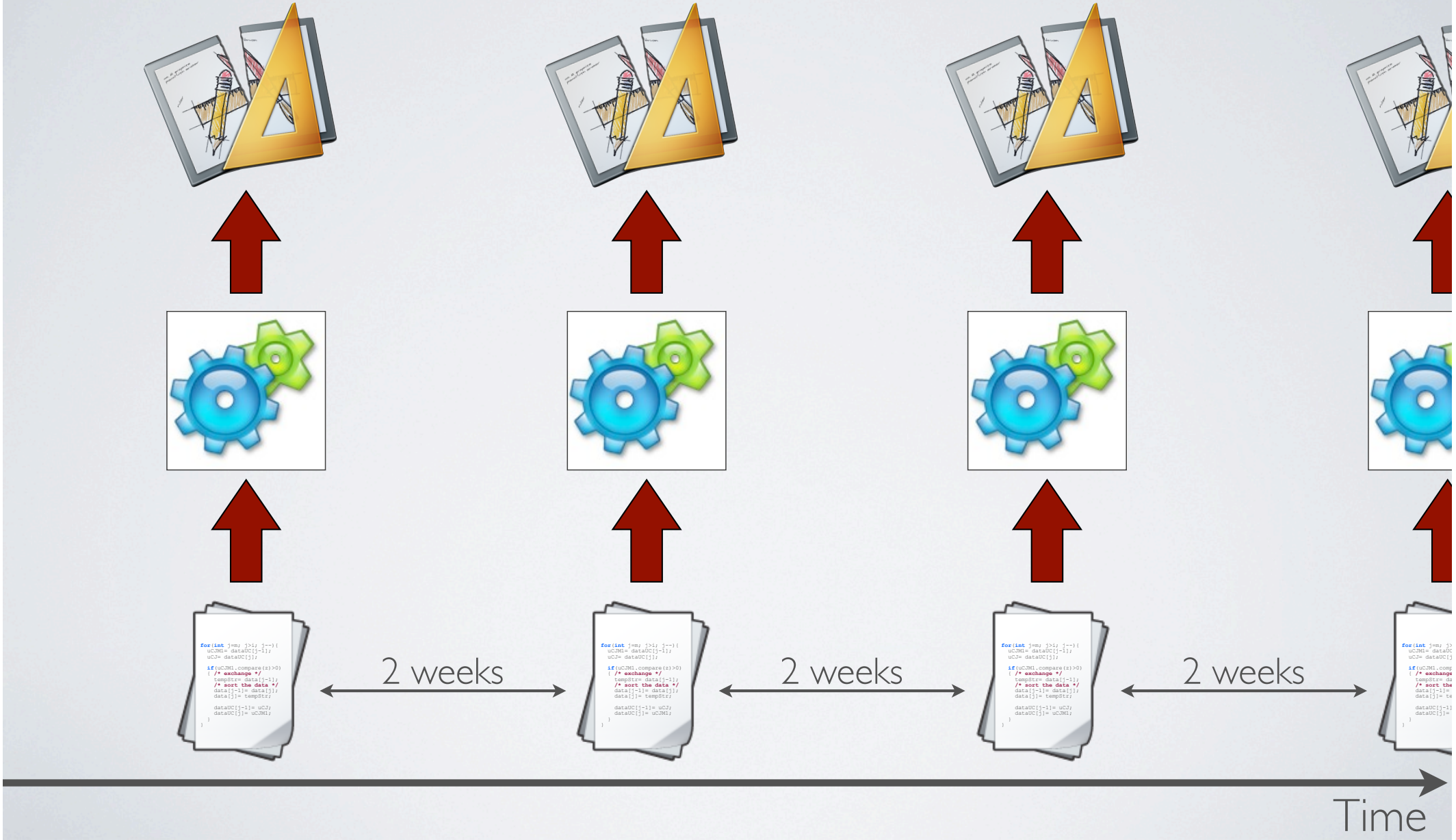
Source code



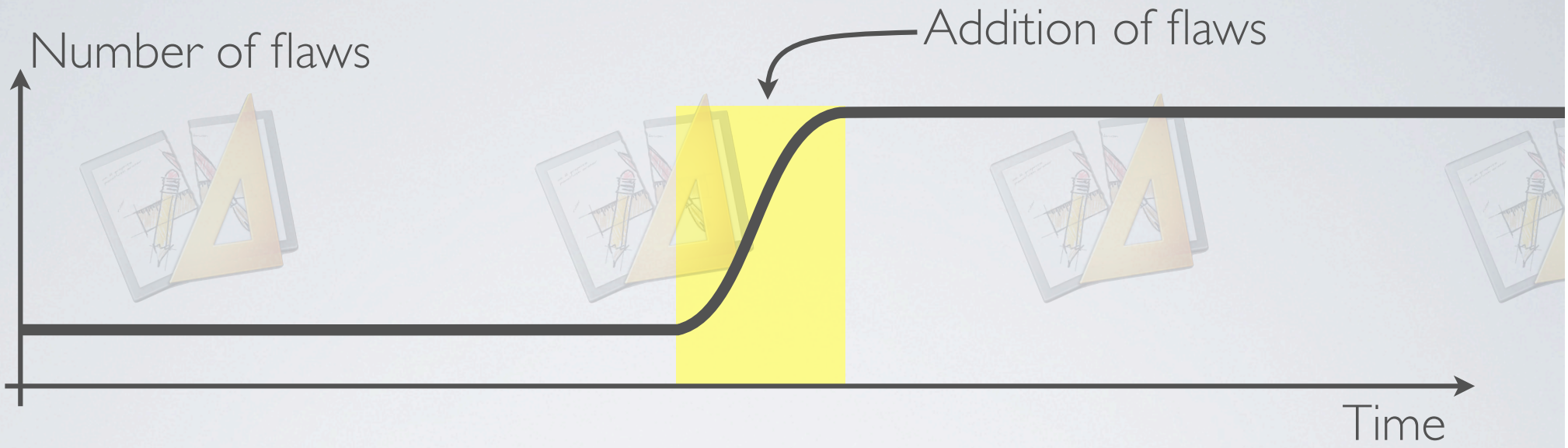
```
for(int j=m; j>i; j--){
  uCJM1= dataUC[j-1];
  uCJ= dataUC[j];

  if(uCJM1.compare(z)>0)
  { /* exchange */
    tempStr= data[j-1];
    /* sort the data */
    data[j-1]= data[j];
    data[j]= tempStr;

    dataUC[j-1]= uCJ;
    dataUC[j]= uCJM1;
  }
}
```

Lanza and Marinescu, *Object-Oriented Metrics in Practice*,Springer 2006

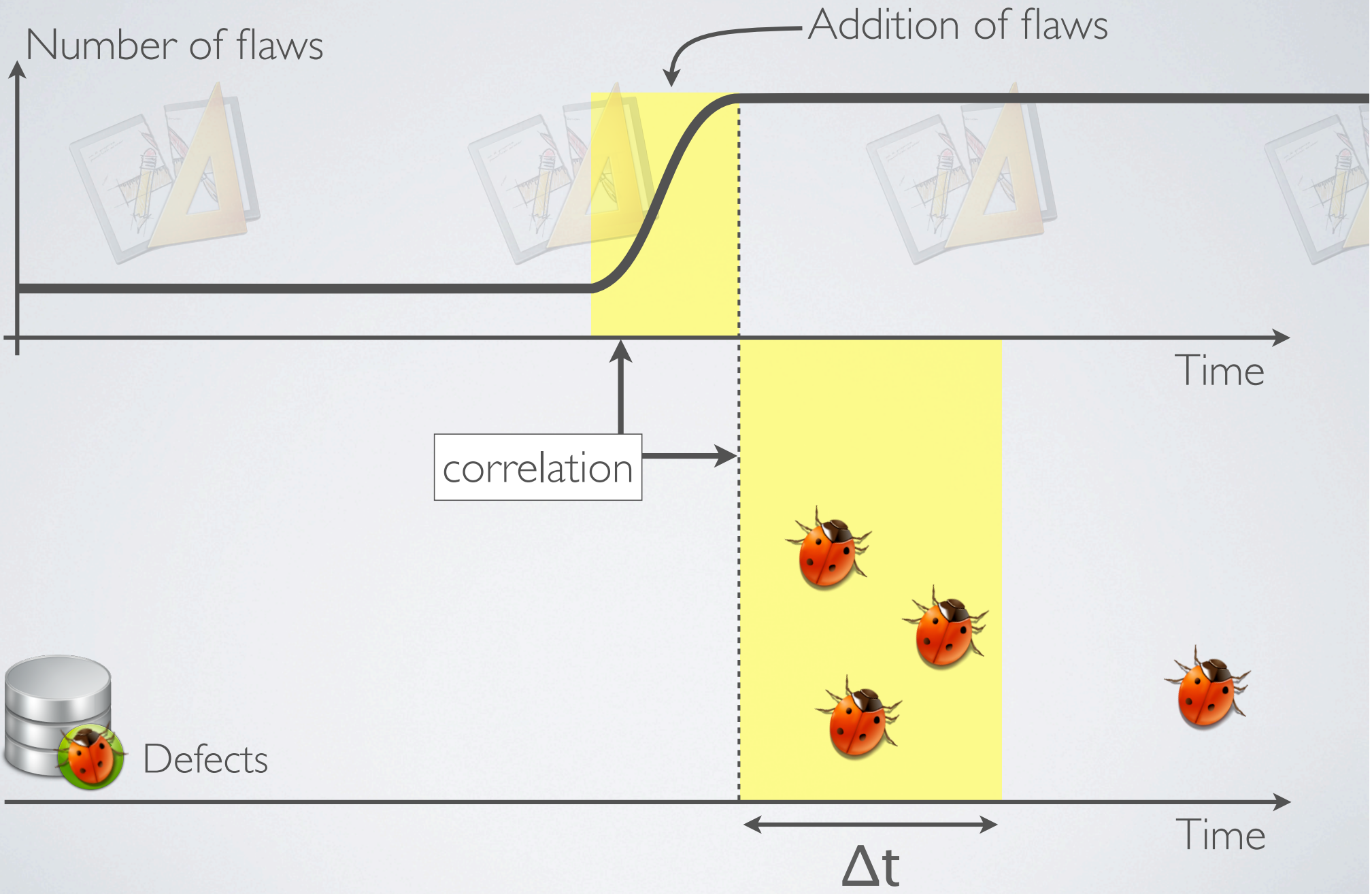
Analyzing design flaw deltas



Analyzing design flaw deltas



Analyzing design flaw deltas



Analyzing design flaw deltas

Flaw presence correlation

0.4+

No flaw correlates more than others consistently across systems

0.6+

Flaw addition correlation

Empirical studies

- Goal: Analyze software repositories data to:
  - empirically validate/invalidate common wisdom
  - investigate a particular phenomenon
  - provide actionable suggestions based on empirical evidence

Defect prediction

Change analysis and prediction

Empirical studies

Expertise & bug triaging

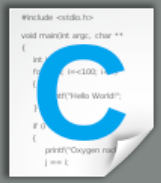
Visual evolution analysis

Human factors

Expertise & bug triaging

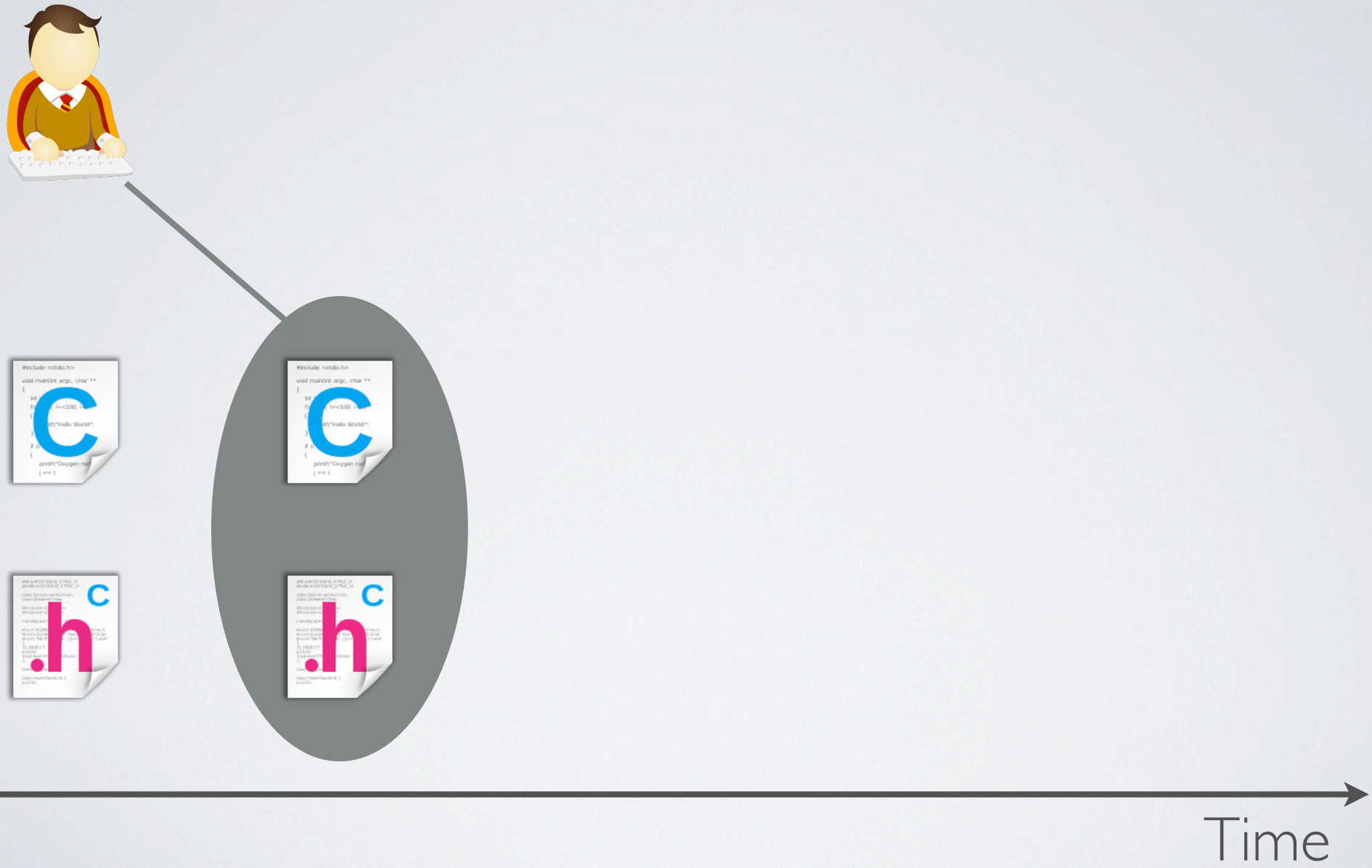
- Goal: Study historical data to:
  - find experts of a particular software artifact
  - support bug triaging

Finding experts for a software artifact

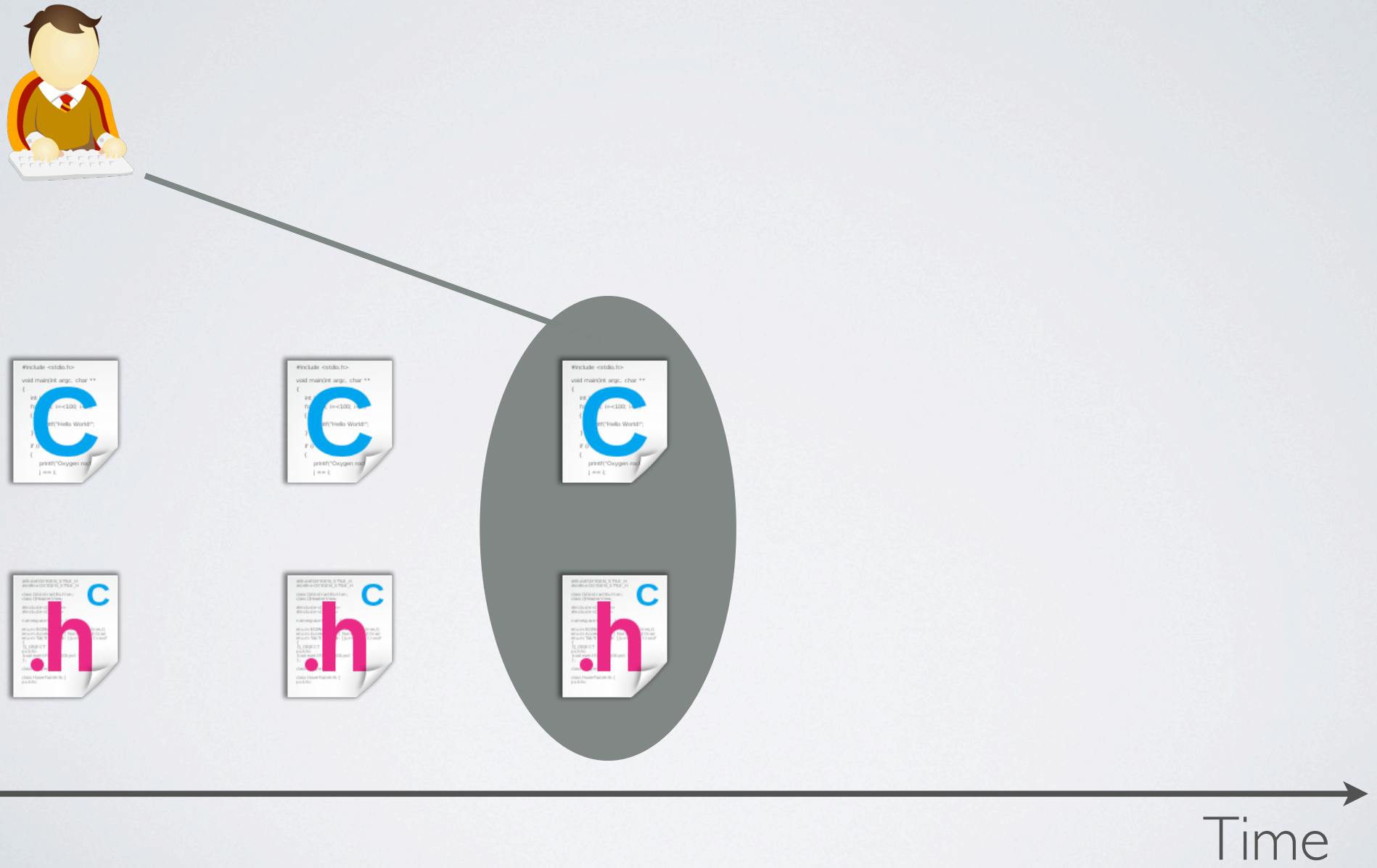


Time

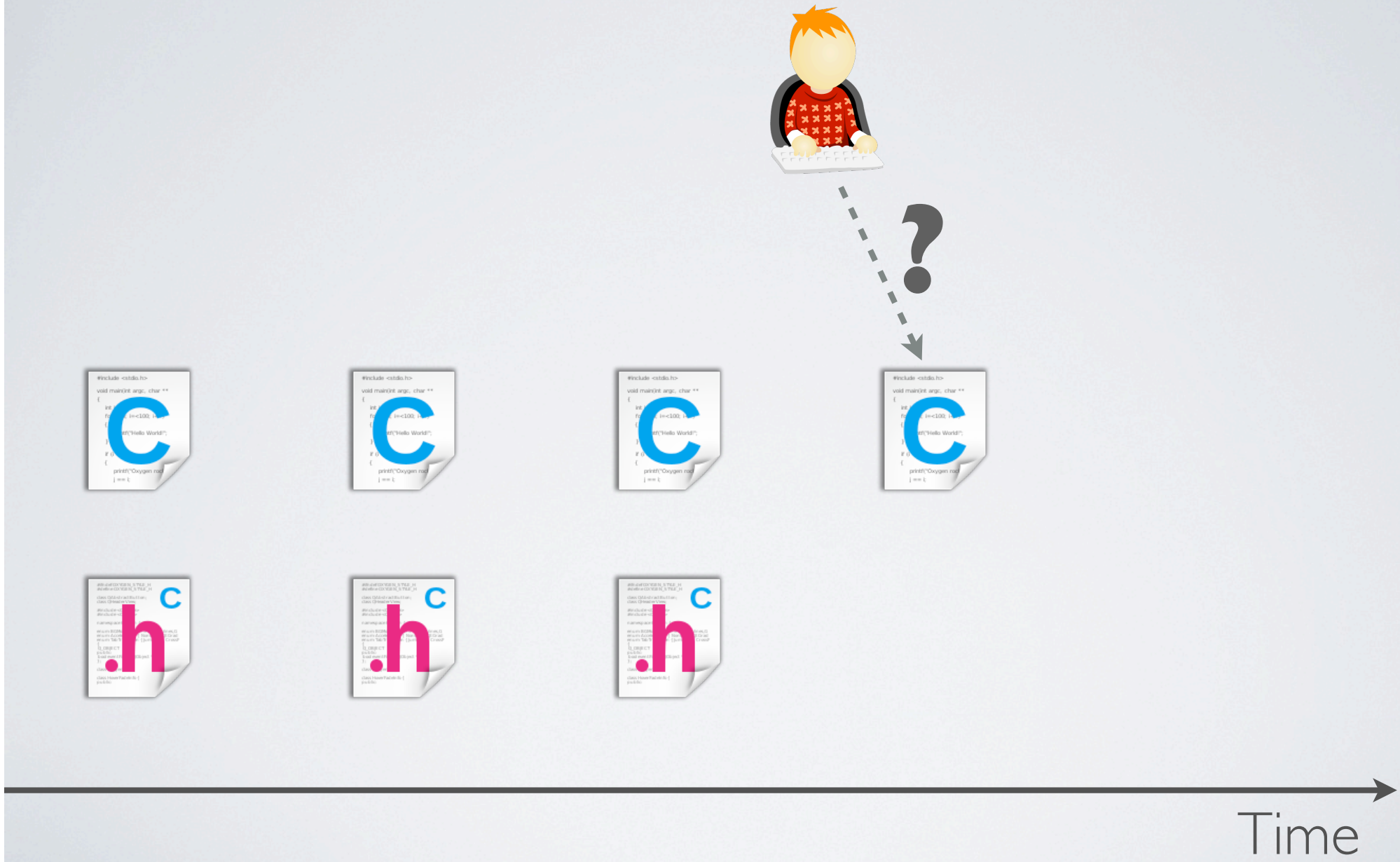
Finding experts for a software artifact



Finding experts for a software artifact



Finding experts for a software artifact



Finding experts for a software artifact

Time

The line 10 rule

- The most basic and famous heuristics for expertise detection
- Look into the SCM for the last person who modified an artifact
- This person is the one that has the code "freshest" in mind
- This heuristic is considered good enough to find help about a piece of code
- The name stems from an SCM that stores the author who did the commit in line 10 of the commit's log message

The Expertise Recommender (ER)

- The ER combines the line 10 rule with "tech support heuristic" mined from a support database

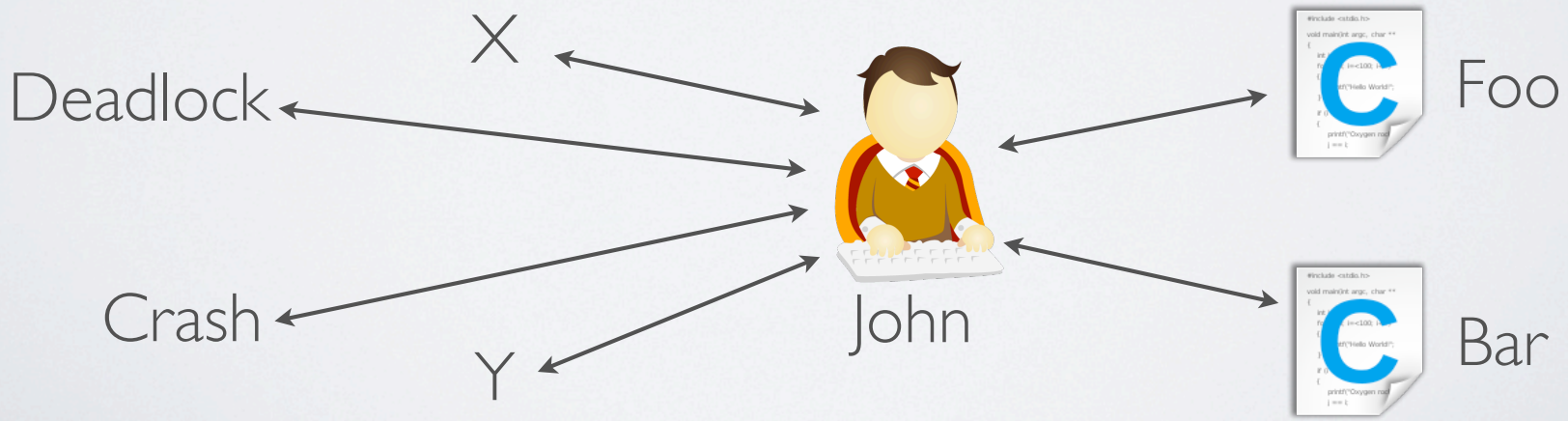
customer	problem symptoms	software modules	solved by
X	Crash	Foo	John
Y	Deadlock	Bar	John

McDonald and Ackerman, *Expertise recommender: a flexible recommendation system and architecture*, CSCW 2000

The Expertise Recommender (ER)

- The ER combines the line 10 rule with "tech support heuristic" mined from a support database

customer	problem symptoms	software modules	solved by
X	Crash	Foo	John
Y	Deadlock	Bar	John



Providing recommendation with ER

- Given a problem symptom, it looks for similar problems and recommend people who solved them in the past
- The ER system enriches the line 10 and tech support heuristics with social network information
  - it recommends only known people
  - it recommends only people from the same department

The Expertise Browser (ExB)

- Main features
  - extraction of developers' experience distribution over a system (i.e., highly specialized vs broad)
  - identification of where contributions occurred
  - discrimination of various kind of experience (e.g., tool, language, release)
- The ExB is based on basic units of experience called experience atoms (EA)

Mockus and Herbsleb, *Expertise browser: a quantitative approach to identifying expertise*, ICSE 2002

Experience Atoms (EA)

- EAs are created by mining the SCM for the author of each file revision and the changes made to the file
- EAs are associated with multiple domains
  - the file containing a modification
  - the technology used
  - the purpose of the change
  - the release of the software
- By counting EAs, for each domain in question, the ExB computes the experience in the area in question

The Emergent Expertise Locator (EEL)

- Improvement of previous approaches based on co-change information
- EEL extracts the emergent team structure from the use of the files by developers
- It improves the Expertise Browser approach by considering the relationship between files changed together
- Based on the Expertise Matrix

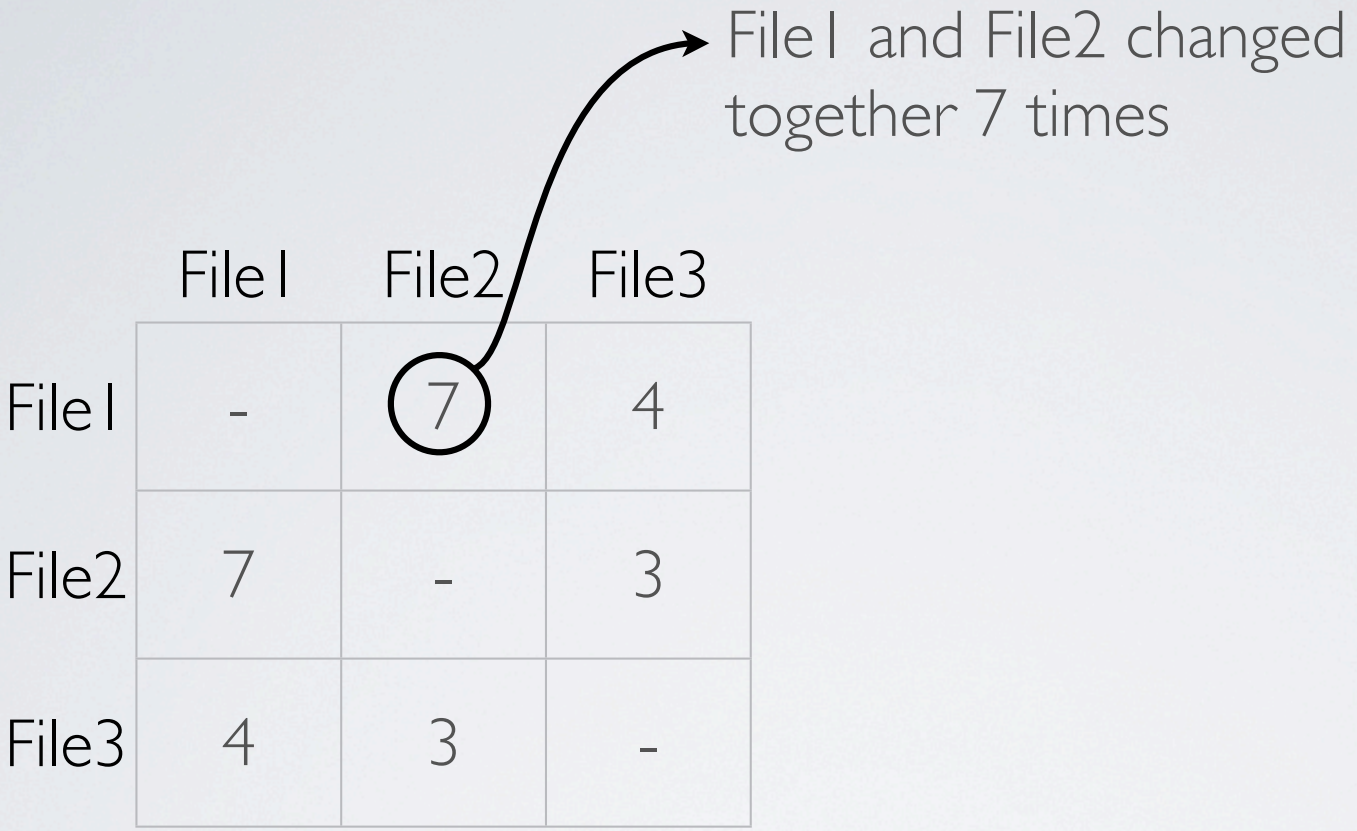
Minto and Murphy, *Recommending emergent teams*, MSR 2007

The Expertise Matrix

	File1	File2	File3
File1	-	7	4
File2	7	-	3
File3	4	3	-

File dependency matrix (F_D)

The Expertise Matrix



	File1	File2	File3
File1	-	7	4
File2	7	-	3
File3	4	3	-

File dependency matrix (F_D)

The Expertise Matrix

	File1	File2	File3
File1	-	7	4
File2	7	-	3
File3	4	3	-

File dependency matrix (F_D)

	File1	File2	File3
Author1	3	5	2
Author2	7	4	10

File authorship matrix (F_A)

The Expertise Matrix

Author1 changed file2 twice

	File1	File2	File3
File1	-	7	4
File2	7	-	3
File3	4	3	-

File dependency matrix (F_D)

	File1	File2	File3
Author1	3	5	2
Author2	7	4	10

File authorship matrix (F_A)

The Expertise Matrix

$$C = (F_A F_D) F_A^T$$

	Author1	Author2	Author3
Author1	-	3	2
Author2	3	-	1
Author3	2	1	-

Amount of expertise that author2 has to author3

Usage expertise

- What do we do for files with little or no history?
- Underlying idea: developers gain expertise not only by changing methods, but also by calling (using) them
- Schuler and Zimmermann mined CVS commits data to extract (by comparing pairs of versions):
  - changed methods (implementation expertise)
  - inserted methods calls (usage expertise)
    - getter/setter methods and Java API calls are ignored

Schuler and Zimmermann, *Mining Usage Expertise from Version Archives*, MSR 2007

Building expertise profiles

- Implementation and usage expertise form expertise profile

Ten most frequently changed		Ten most frequently used	
`createPartControl`	185	`addSelectionListener`	72
`aboutToStart`	163	`openError`	57
`createControl`	148	`addModifyListener`	35
`rerunTest`	143	`refreshStatus`	31
`menuAboutToShow`	142	`addTest`	29
`testFailed`	136	`worked`	26
`testReran`	117	`asyncExec`	24
`testRunEnded`	114	`handleFieldChanged`	24
`showFailure`	113	`postRefresh`	24
`endTest`	109	`findType`	21

Profile of Erich Gamma

Example usage of expertise profiles

- Which ECLIPSE developers have experience in using the SQL part of the JAVA SDK?

Q = methods that are contained in the java.sql package

U_d = methods most frequently used by developer d

Developer d	$\lvert U_d \cap Q \rvert$
aweinand	18
c[illegible]	11
dj[illegible]	10
bb[illegible]owski	9
othomann	8
dorme	7
dbaeumer	7

best match

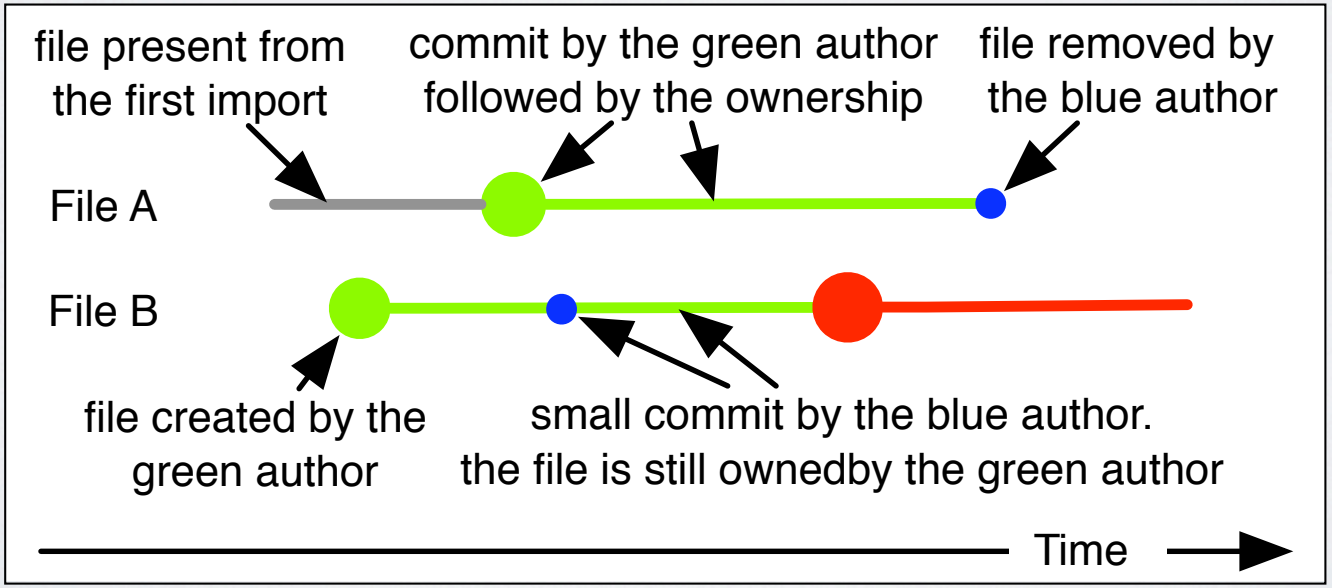
Refining and visualizing code ownership

- Line ownership: a developer owns a line of code if he was the last one that committed a change to that line
- File ownership: a developer owns a file if he/she owns the largest part of it (in terms of lines)

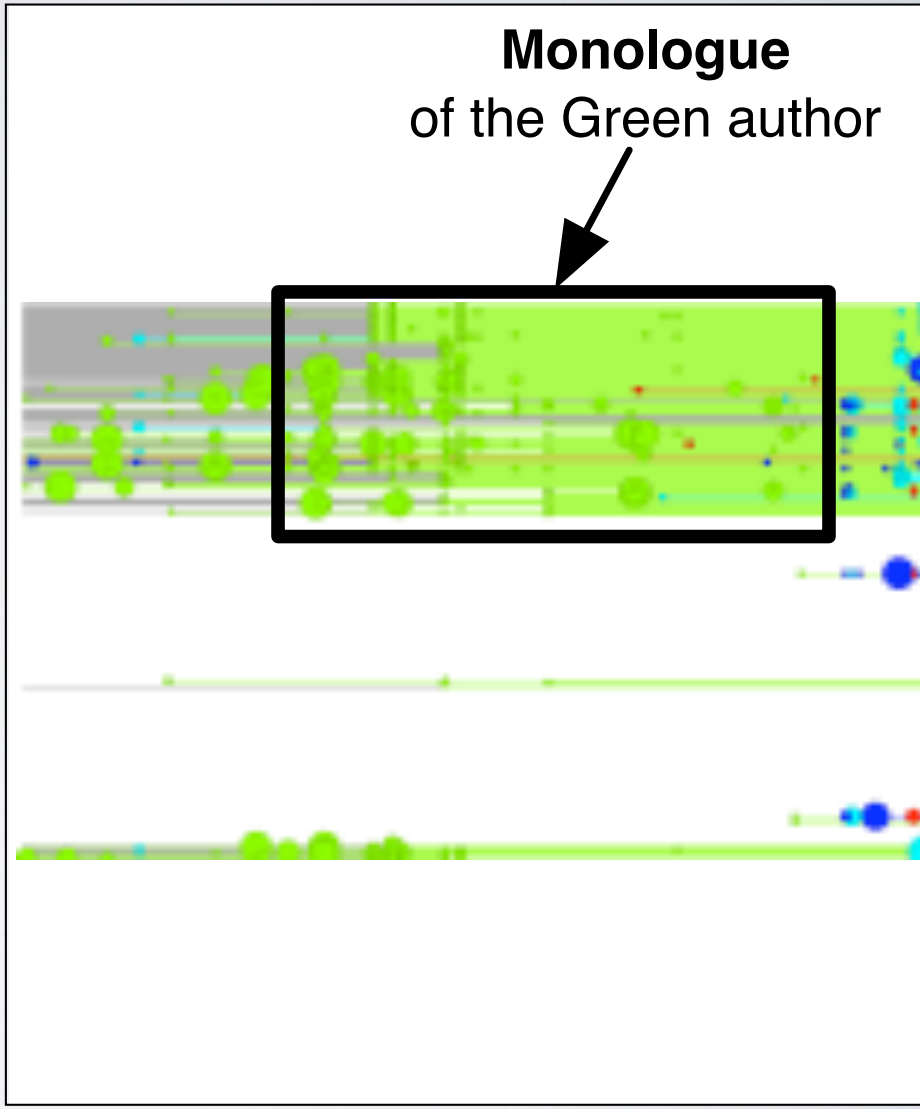
Gîrba et al., *How Developers Drive Software Evolution*, IWPSE 2005

Refining and visualizing code ownership

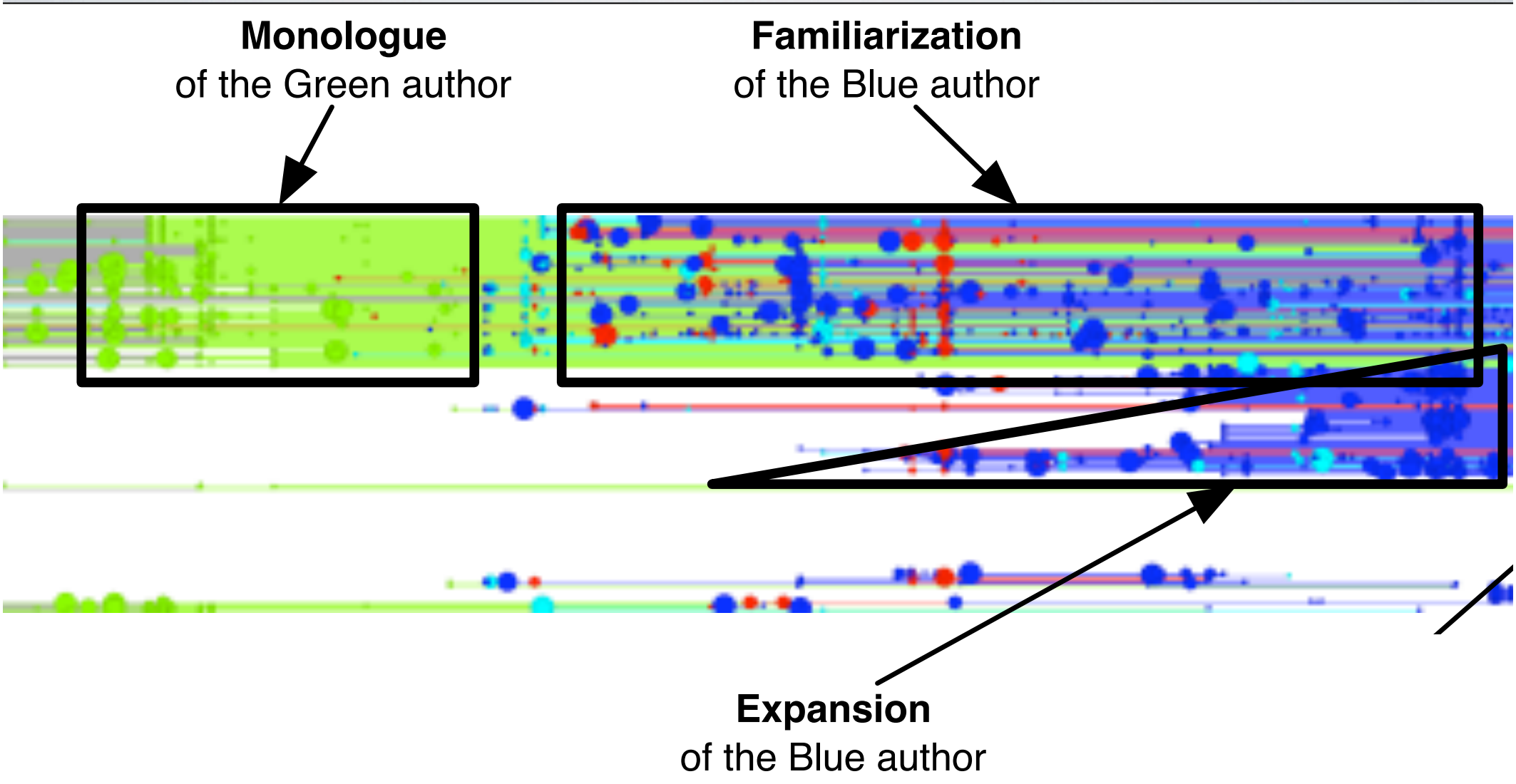
- Line ownership: a developer owns a line of code if he was the last one that committed a change to that line
- File ownership: a developer owns a file if he/she owns the largest part of it (in terms of lines)



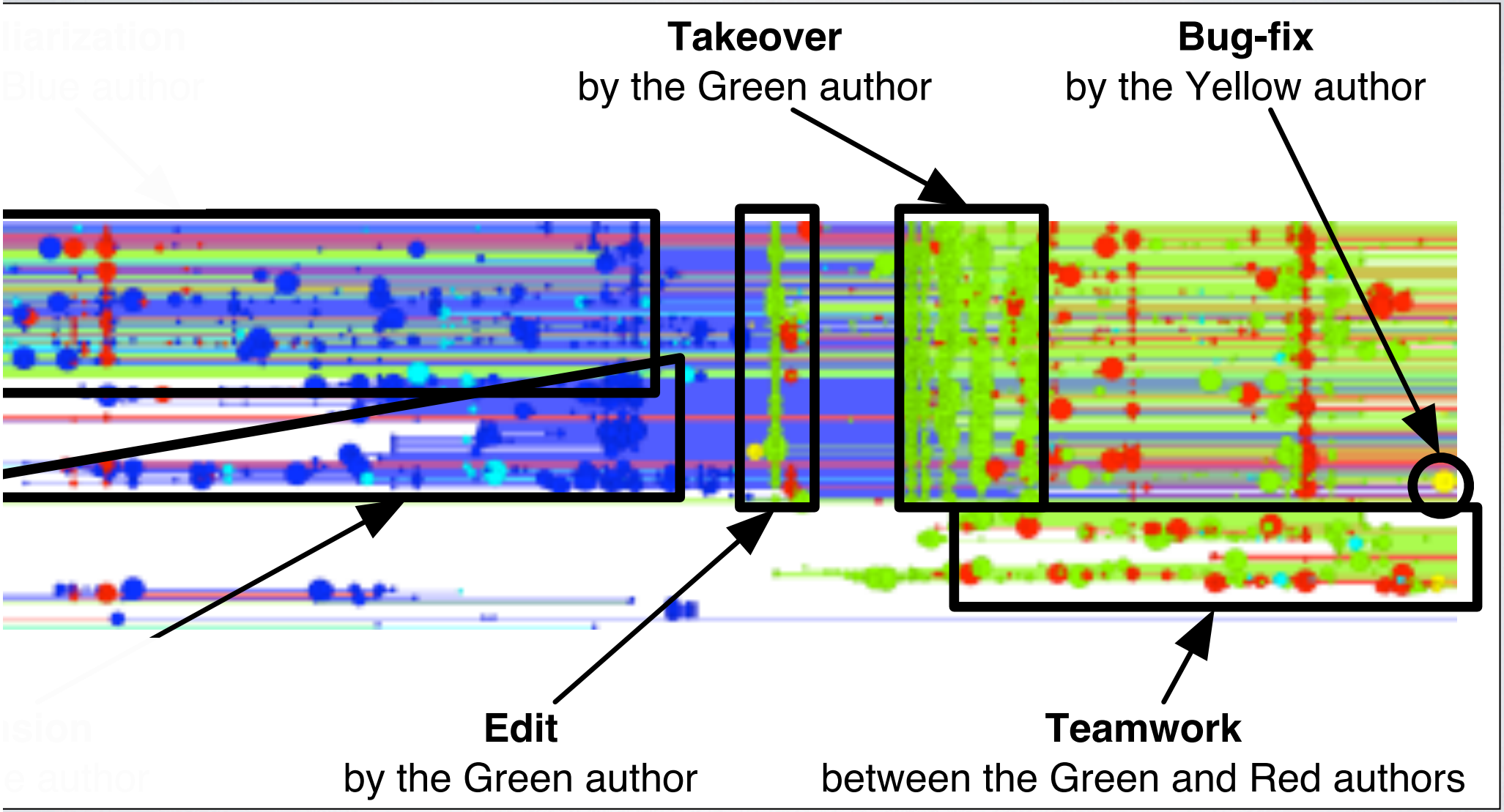
Visualization based development patterns



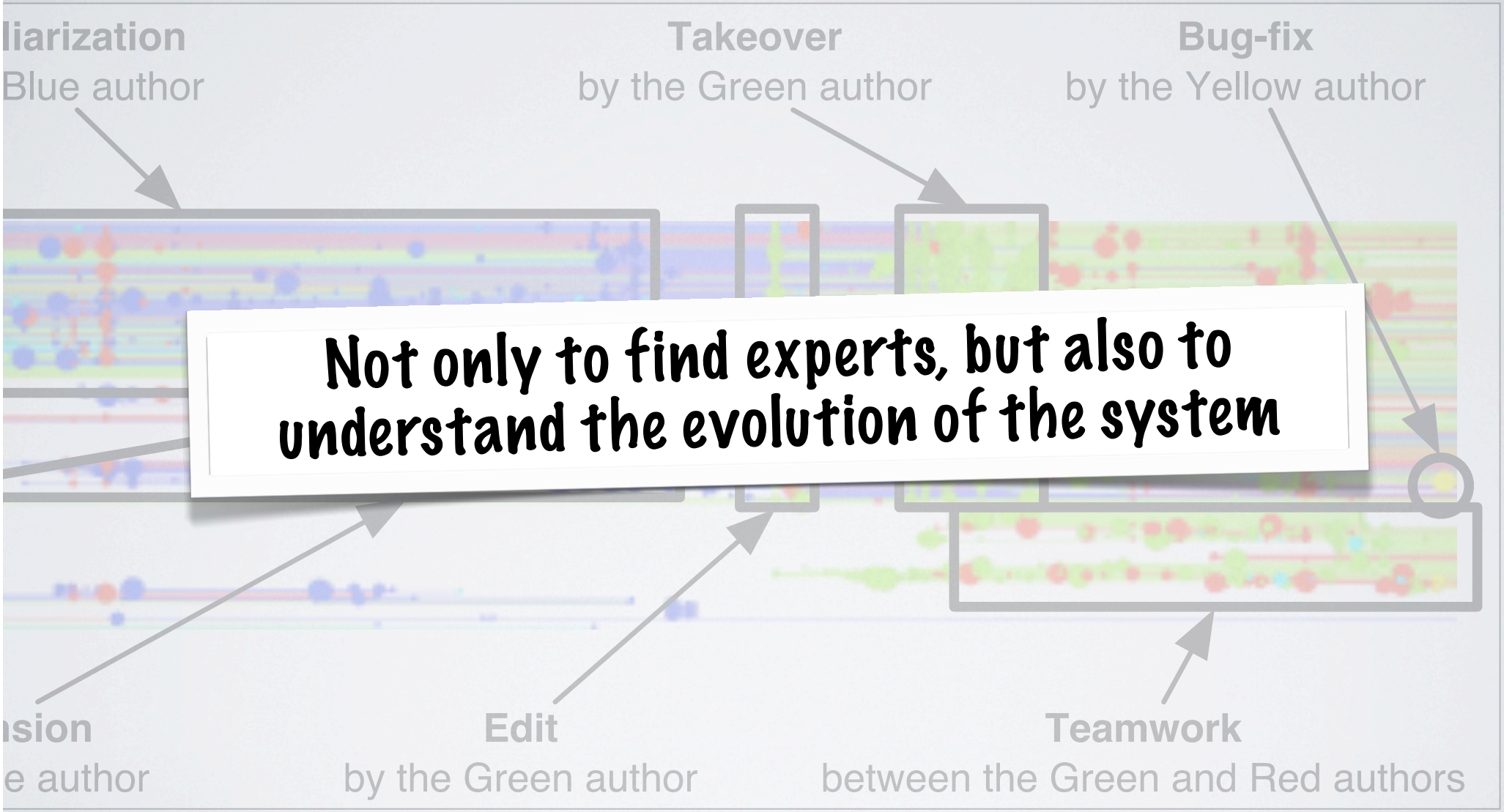
Visualization based development patterns

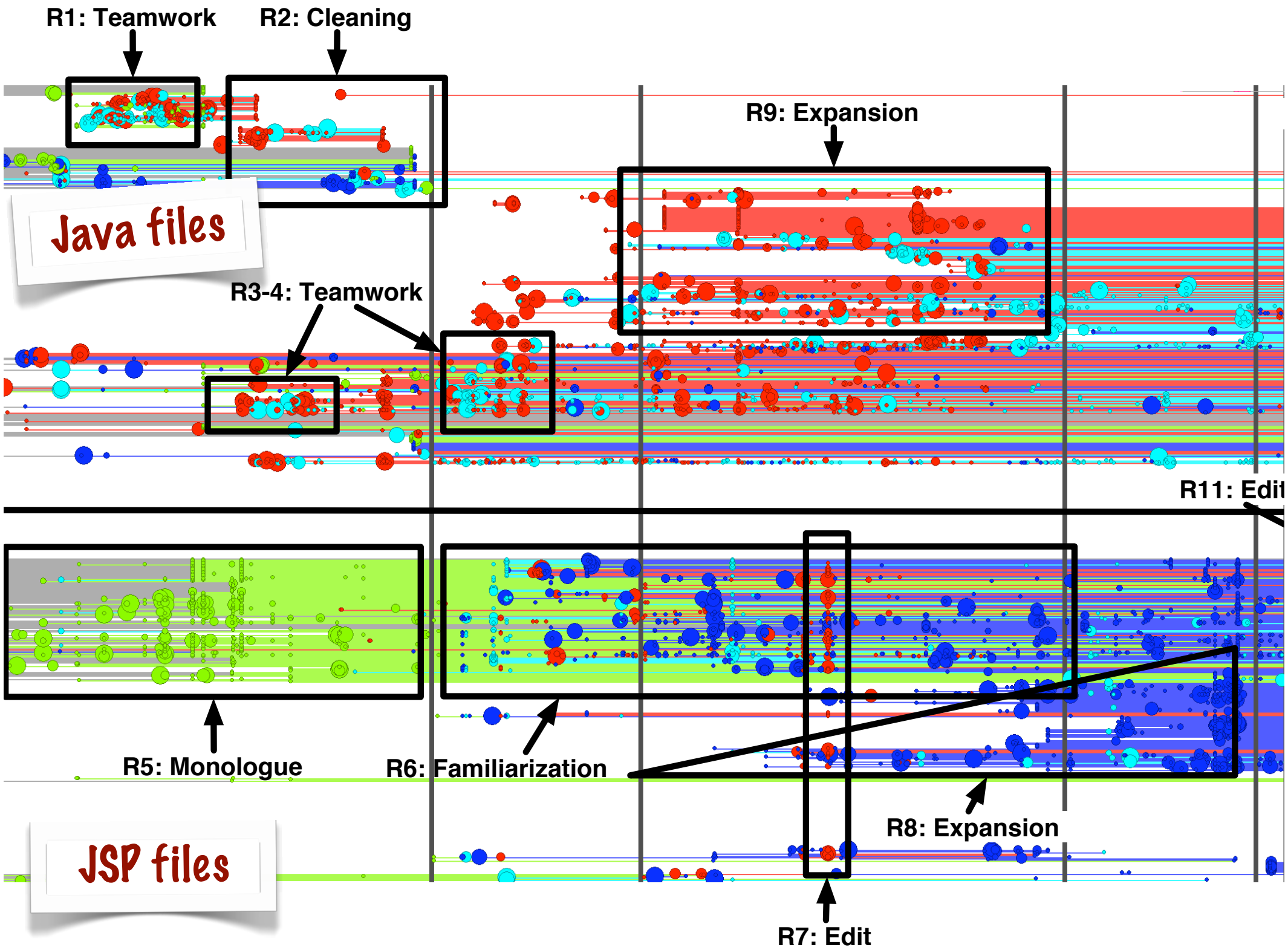


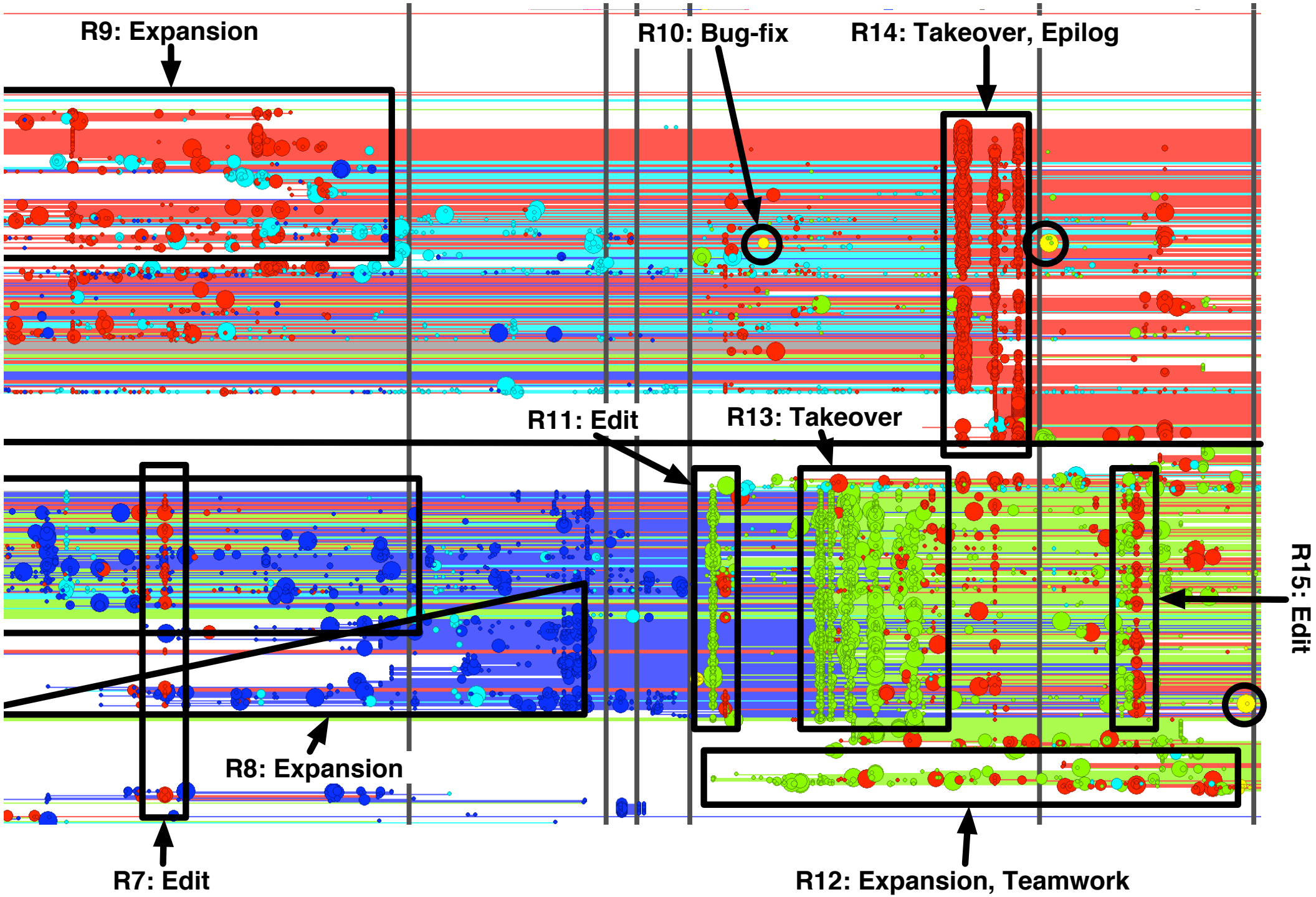
Visualization based development patterns

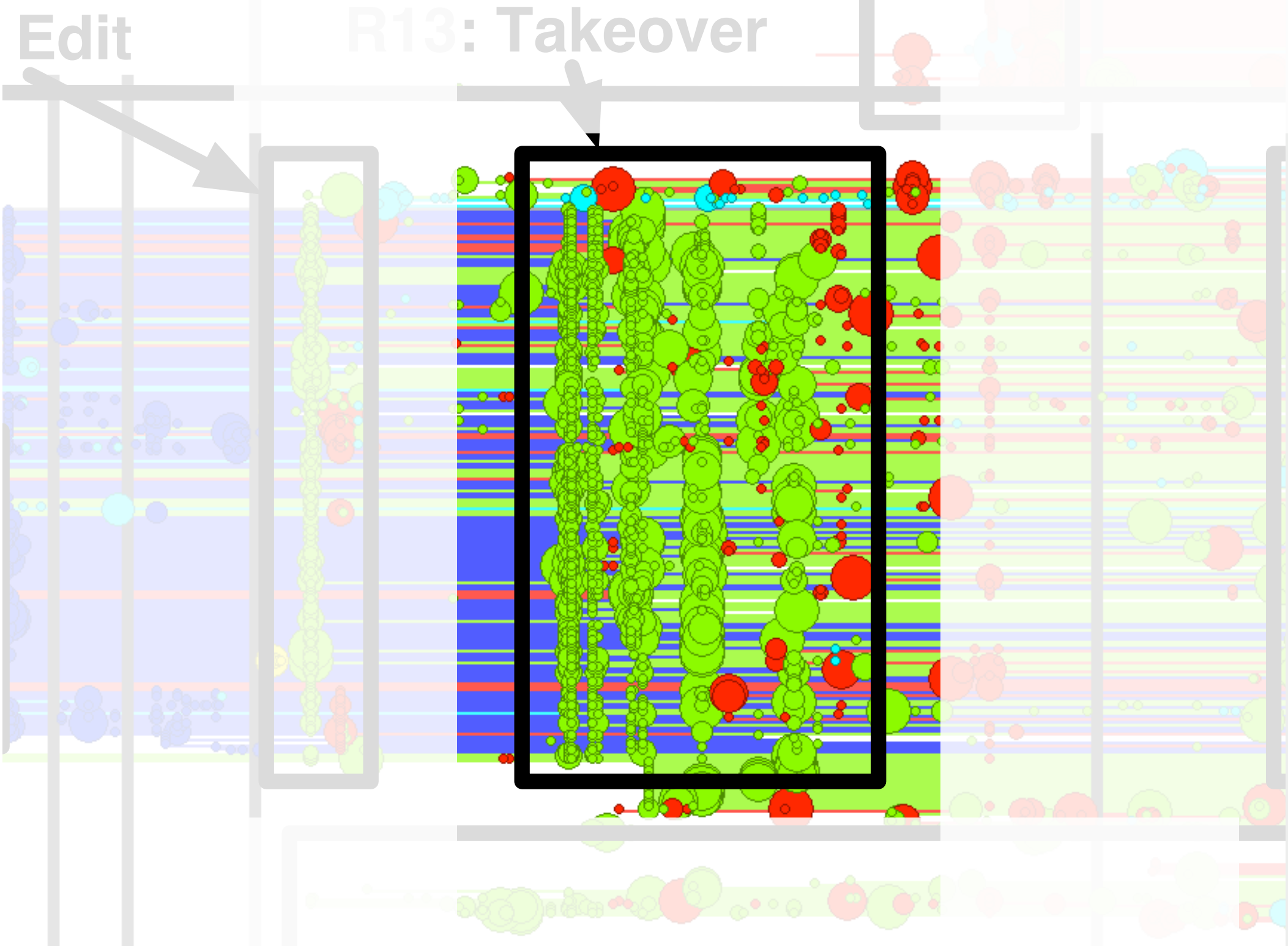


Visualization based development patterns




R1: Teamwork
R2: Cleaning
Java files
R9: Expansion
R3-4: Teamwork
R11: Edit
R5: Monologue
R6: Familiarization
R8: Expansion
JSP files
R7: Edit


R9: Expansion
R10: Bug-fix
R14: Takeover, Epilog
R11: Edit
R13: Takeover
R15: Edit
R8: Expansion
R7: Edit
R12: Expansion, Teamwork

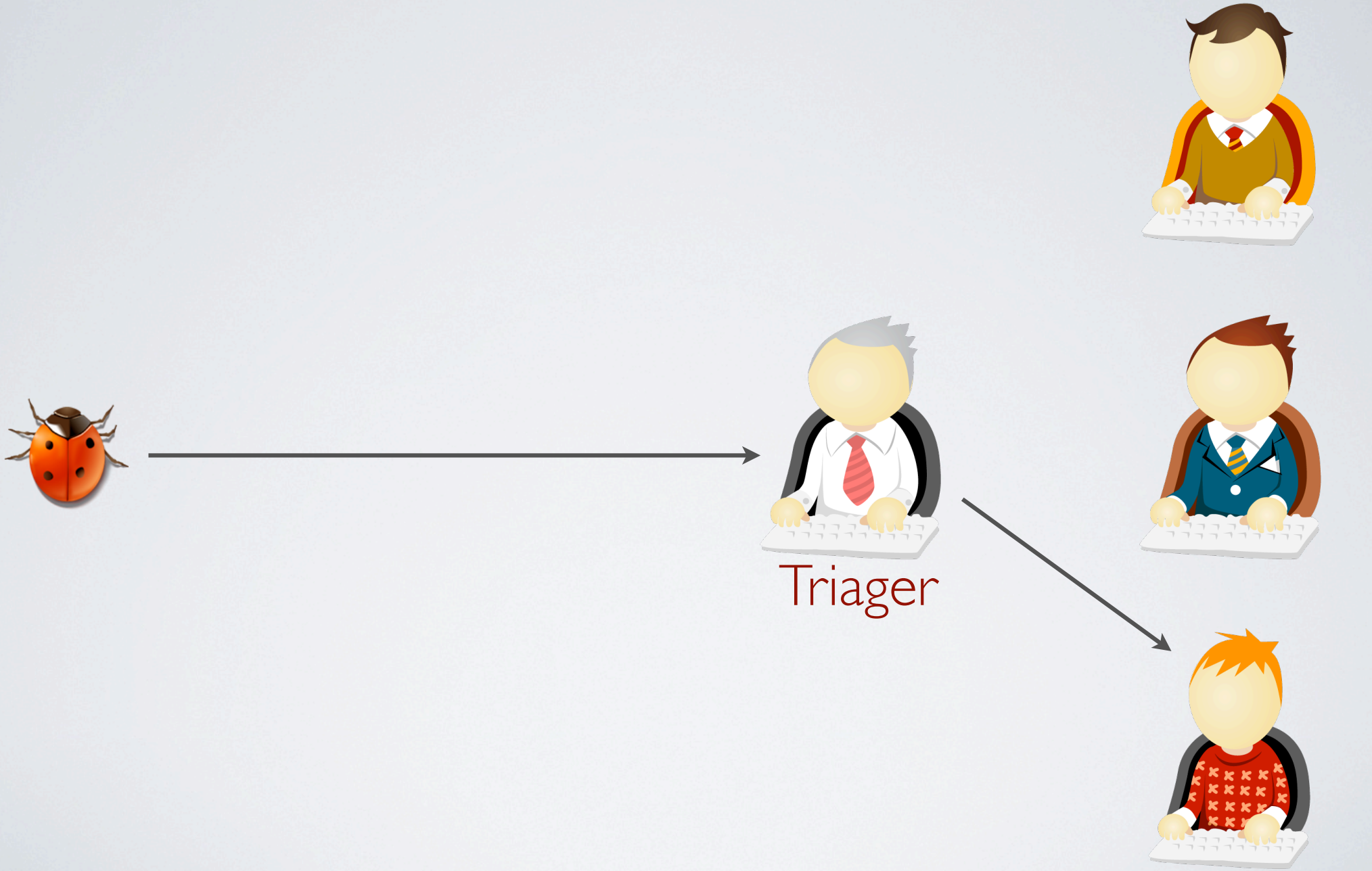

Edit
R13: Takeover

Bug triaging

Developers

Triager

Bug triaging



Semi-automated bug triaging

Developers

→

Bug triaging algorithm

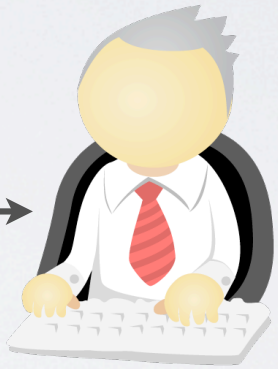
Triager

Semi-automated bug triaging

Developers

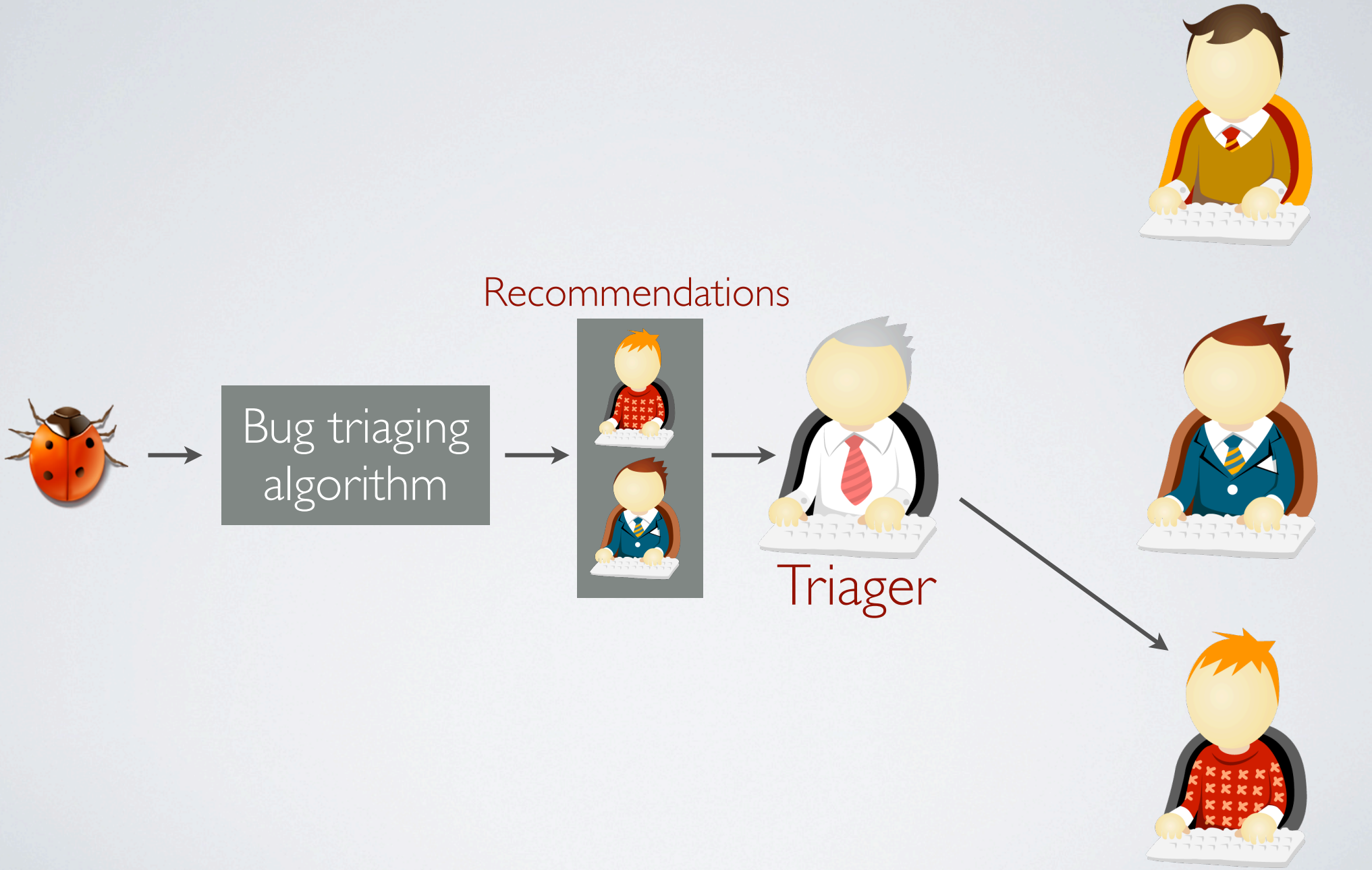
Bug triaging algorithm

Recommendations



Triager

Semi-automated bug triaging



Semi-automated bug triaging

- Anvik et al. proposed a 4 steps approach

1. Summary and full text description of bug report are extracted, preprocessed (e.g. removing stop-words) and a feature vector indicating the frequency of the terms in the text is created

2. For each bug report, the developer who fixed it is detected

    - This does not always correspond to the "assigned-to" field of the bug report (e.g. new and unassigned bugs are assigned to a default email address, for bugs with trivial resolution the "assign-to" is often never changed)

    - A series of project specific heuristics are used for detecting the "assigned-to"

        - For example, if a report is resolved as FIXED, it was fixed by whoever submitted the last approved patch

Anvik et al., *Who should fix this bug?*, ICSE 2006

Semi-automated bug triaging

- Anvik et al. proposed a 4 steps approach

3. A set of bug reports is selected for training the model, filtering out reports for which
   - the heuristics cannot provide a useful "assigned-to"
   - the developer is no longer involved in the project, or he/she has fixed too few bugs
4. Apply machine learning algorithms on the training set
   - Support Vector Machines (SVM), Naïve Bayes and decition trees (C4.5) are tested
   - SVM provides the best results

Anvik et al., *Who should fix this bug?*, ICSE 2006

Semi-automated bug triaging

- Anvik et al. proposed a 4 steps approach

3. A set of bug reports is selected for training the model, filtering out reports for which
   - the heuristics cannot provide a useful "assigned-to"

A bug is assigned to a developer who fixed bugs with similar properties (e.g. description)

4. Apply machine learning algorithms on the training set
   - Support Vector Machines (SVM), Naïve Bayes and decition trees (C4.5) are tested
   - SVM provides the best results

Anvik et al., *Who should fix this bug?*, ICSE 2006

Triaging results

- Approach evaluated on bug reports from Firefox and Eclipse
  - Precision: ~50% with maximum 64% on Firefox
  - Recall: ~10% with maximum 16% on Eclipse
- Approach tested on GCC
  - Precision only 6%... why?
    - Unbalanced data: one developer dominates the bug resolution activity
    - The heuristics to detected the "assigned-to" are not accurate enough

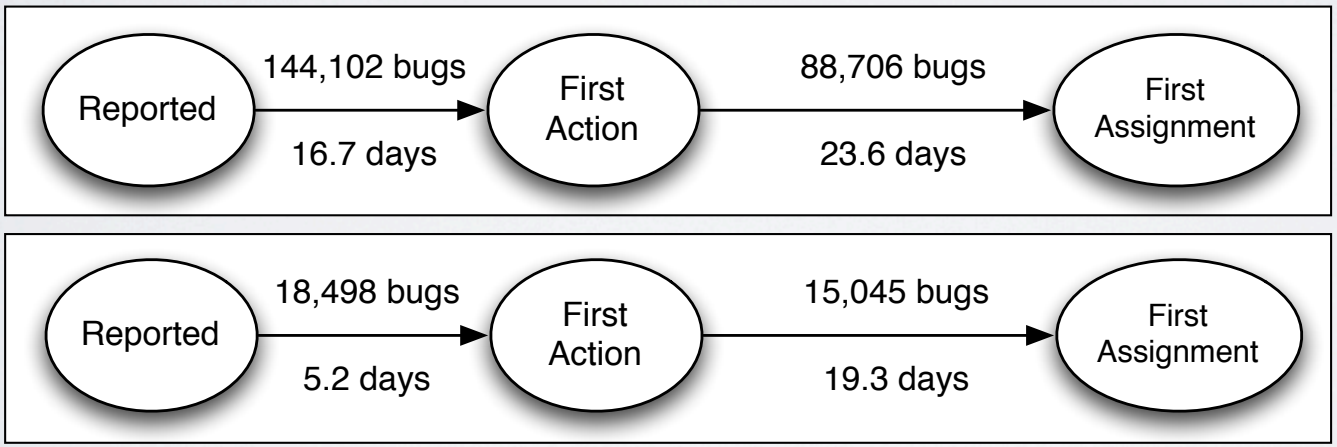
Improving Bug Triage with Bug Tossing Graphs

- Tossing: reassignment of a bug
- Work motivated by empirical observations on 445,000 bug reports from Mozilla and Eclipse

Jeong et al., *Improving bug triage with bug tossing graphs*, ESEC/FSE 2009

Improving Bug Triage with Bug Tossing Graphs

- Tossing: reassignment of a bug
- Work motivated by empirical observations on 445,000 bug reports from Mozilla and Eclipse
  - Assigning bugs takes long



All Eclipse bugs

Eclipse verified bugs

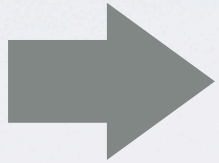
  - 37% (Mozilla) - 44% (Eclipse) of bugs are tossed

Modeling tossing graph with Markov chain

- Assumption: it does not matter for tossing decisions whether someone received a bug from Alice or Bob

$A \to B \to C \to D$
$A \to C \to D \to E$
$C \to E \to A \to F \to D$

Tossing path



$A \to B\ (1)$, $B \to C\ (1)$,
$C \to D\ (2)$, $A \to C\ (1)$,
$D \to E\ (1)$, $C \to E\ (1)$,
$E \to A\ (1)$, $A \to F\ (1)$,
$F \to D\ (1)$

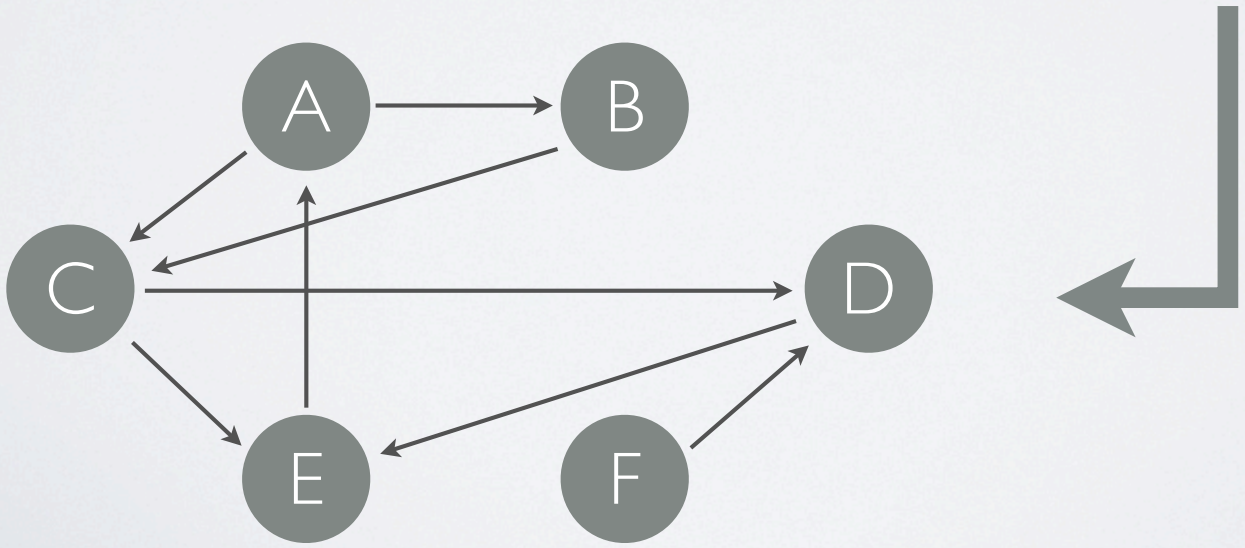
Modeling tossing graph with Markov chain

- Assumption: it does not matter for tossing decisions whether someone received a bug from Alice or Bob

$$A \to B \to C \to D$$
$$A \to C \to D \to E$$
$$C \to E \to A \to F \to D$$

Tossing path

$$A \to B\,(1), B \to C\,(1),$$
$$C \to D\,(2), A \to C\,(1),$$
$$D \to E\,(1), C \to E\,(1),$$
$$E \to A\,(1), A \to F\,(1),$$
$$F \to D\,(1)$$



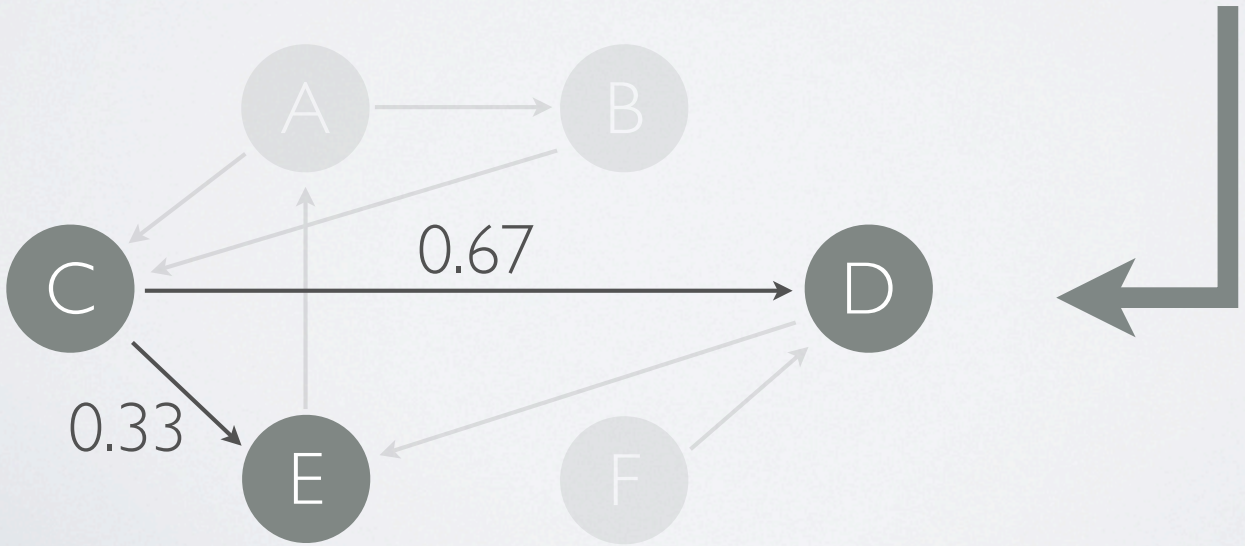
Modeling tossing graph with Markov chain

- Assumption: it does not matter for tossing decisions whether someone received a bug from Alice or Bob

$$A \to B \to C \to D$$
$$A \to C \to D \to E$$
$$C \to E \to A \to F \to D$$

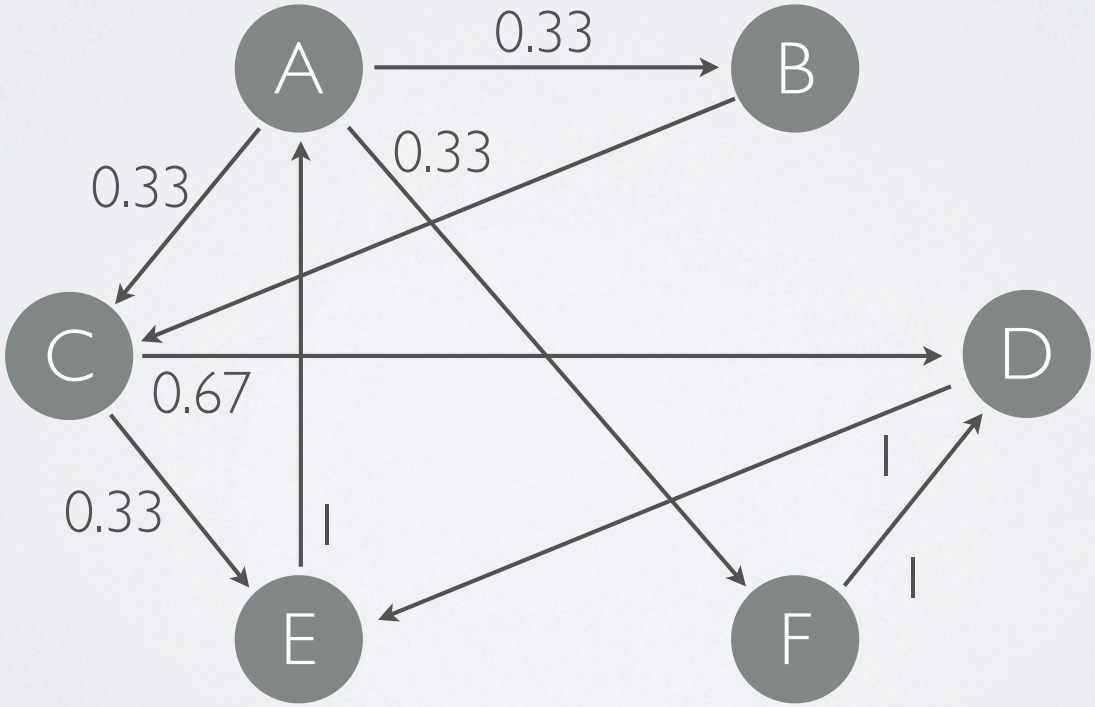
Tossing path

$$A \to B\,(1),\ B \to C\,(1),$$
$$C \to D\,(2),\ A \to C\,(1),$$
$$D \to E\,(1),\ C \to E\,(1),$$
$$E \to A\,(1),\ A \to F\,(1),$$
$$F \to D\,(1)$$



Minimum support and transaction probability

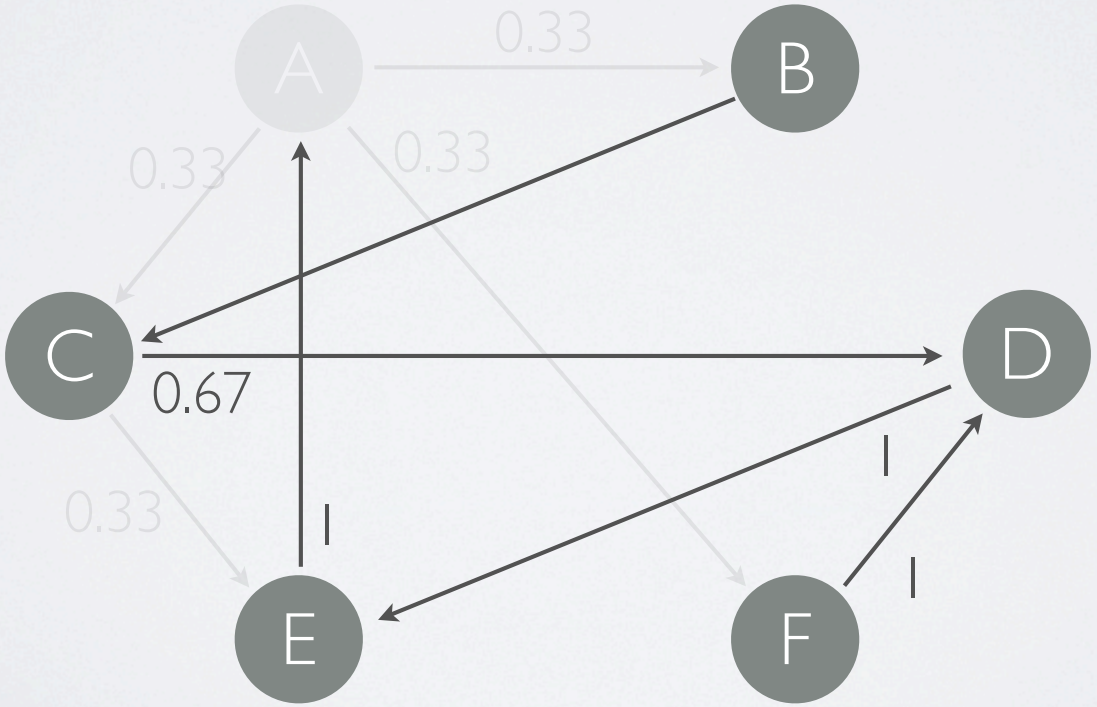
$A \rightarrow B\,(1), B \rightarrow C\,(1),$
$C \rightarrow D\,(2), A \rightarrow C\,(1),$
$D \rightarrow E\,(1), C \rightarrow E\,(1),$
$E \rightarrow A\,(1), A \rightarrow F\,(1),$
$F \rightarrow D\,(1)$



Minimum support and transaction probability

$A \rightarrow B\ (1),\ B \rightarrow C\ (1),$
$C \rightarrow D\ (2),\ A \rightarrow C\ (1),$
$D \rightarrow E\ (1),\ C \rightarrow E\ (1),$
$E \rightarrow A\ (1),\ A \rightarrow F\ (1),$
$F \rightarrow D\ (1)$

Minimum probability: 0.5



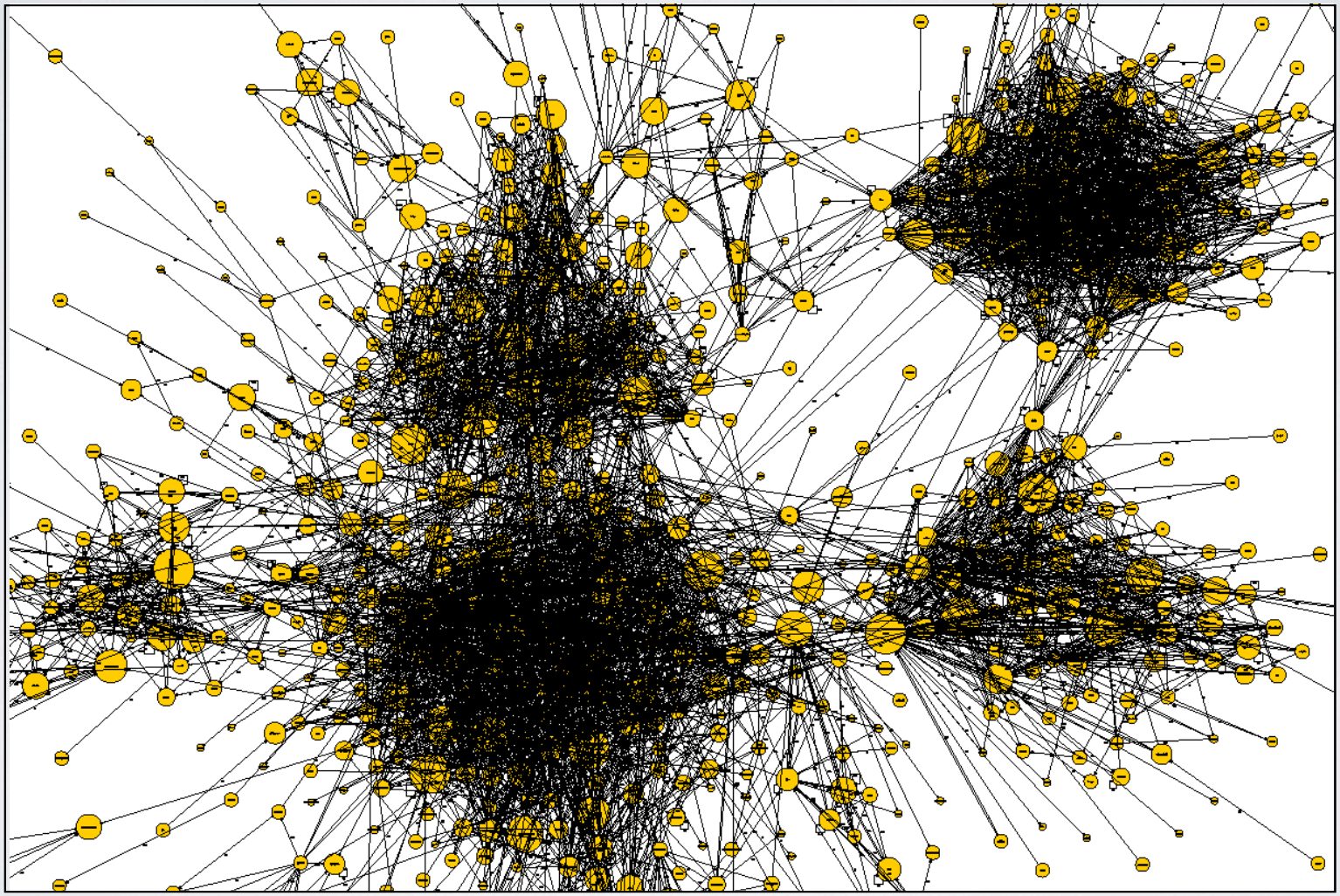
Minimum support and transaction probability

$A \rightarrow B\ (1), B \rightarrow C\ (1),$
$C \rightarrow D\ (2), A \rightarrow C\ (1),$
$D \rightarrow E\ (1), C \rightarrow E\ (1),$
$E \rightarrow A\ (1), A \rightarrow F\ (1),$
$F \rightarrow D\ (1)$

Minimum support: 2

Minimum support and transaction probability

Min support: 0.5 Min transaction probability: 0.5



Eclipse

Improving bug triaging with tossing information

Bug triaging algorithm

Suggested developers

1 robert.elves
2 mik.kersten
3 wuamy
4 susan_franklin
5 nitind

Improving bug triaging with tossing information

Bug triaging algorithm

Suggested developers

1 robert.elves
2 mik.kersten
3 wuamy
4 susan_franklin
5 nitind

Strong relationships in the tossing graph

robert.elves ↔ steffen.pingel

mik.kersten ↔ relves

1 robert.elves
2 steffen.pingel
3 mik.kersten
4 relves
5 wuamy

Triaging improvement

Program	ML algorithm	Selection	Accuracy (%)		Improvement
			ML only	ML + tossing graph	
Eclipse	Naïve Bayes	first 2	43.70	44.71	1.01
		first 3	49.87	53.15	3.27
		first 4	56.42	59.95	3.53
		first 5	60.71	63.48	2.77
	Bayesian Network	first 2	57.91	58.29	0.38
		first 3	66.71	68.47	1.76
		first 4	69.47	71.48	2.01
		first 5	75.88	77.14	1.26
Mozilla	Naïve Bayes	first 2	33.41	56.39	22.98
		first 3	45.39	63.82	
		first 4	52		
		first 5	59		
	Bayesian Network	first 2	40		
		first 3	50		
		first 4	55		
		first 5	59.53	70.82	11.29

Improvement up to 23%
Accuracy up to 71%

Expertise & bug triaging

- Goal: Study historical data to:
  - find experts of a particular software artifact
  - support bug triaging

Defect prediction

Change analysis and prediction

Empirical studies

Expertise & bug triaging

Visual evolution analysis

Human factors

Visual Evolution Analysis

- Goal: Visualize evolutionary data to:
  - understanding a software system's evolution
  - support decision making
  - detect design problems (e.g. architecture decay)

Software visualization

> *"the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software"*
>
> *—Stasko et al.*

John Stasko et al. (Eds.), Software Visualization - Programming as a Multimedia Experience. The MIT Press, 1998.

Visualization taxonomy - visualized data

	Data	Goal
Static	information extracted by static analysis of the software	support the understanding of the structure of the system
Dynamic	information derived from instrumenting the execution of a program	support the understanding of the behavior of the system
Evolutionary	information extracted from the evolution of a software system	support the analysis of the causes of problems in the software and the prediction of future changes or problems

Visualization taxonomy - visualized data

	Data	Goal
Static	information extracted by static analysis of the software	support the understanding of the structure of the system
Dynamic	information derived from instrumenting the execution of a program	support the understanding of the behavior of the system
Evolutionary	information extracted from the evolution of a software system	support the analysis of the causes of problems in the software and the prediction of future changes or problems

Visualization taxonomy - abstraction level

Code	visualization of lines of code
Design	visualization of self contained pieces of code, such as classes
Architecture	visualization of system's modules and their inter-relationships

Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows	• SeeSoft	• Augur • CVSscan
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

Visual evolution analysis references

- Code flows - Telea and Auber, *Code Flows: Visualizing structural evolution of source code,* Computer Graphics Forum 2008.
- SeeSoft - Ball and Eick, *Software visualization in the large*, IEEE Computer, 1996
- Augur - Froehlich and Dourish, *Unifying artifacts and activities in a visual tool for distributed software development teams*, ICSE 2004
- CVSscan - Voinea et al., *CVSscan: visualization of code evolution*, SoftVis 2005
- Extension of CodeCrawler (Lanza and Ducasse, TSE 2003) - Gîrba et al., *Characterizing the evolution of class hierarchies*, CSMR 2005
- The Evolution Matrix - Lanza, *The evolution matrix: Recovering software evolution using soft- ware visualization techniques*, IWPSE 2001
- BEAGLE - Tu and Godfrey, *An integrated approach for studying architectural evolution*. IWPC 2002
- CodeCity - Wettel and Lanza, *Visual Exploration of Large-Scale System Evolution*, WCRE 2008
- 3dSoftVis - Gall et al., *Visualizing software release histories: The use of color and third dimension*, ICSM 1999

Visual evolution analysis references

- Mehdi Jazayeri, *On architectural stability and evolution*, Reliable Software Technologies-Ada-Europe 2002
- Revision towers - Taylor and Munro, *Revision towers*, VISSOFT 2002
- Van Rysselberghe and Demeyer, *Studying software evolution information by visualizing the change history*, ICSM 2004
- Evolution Spectrographs - Wu et al., *Exploring software evolution using spectrographs*, WCRE 2004
- Chronia - Gîrba et al., *How developers drive software evolution*, IWPSE 2005
- Evograph - Fischer and Gall, *Evograph: A lightweight approach to evolutionary and structural analysis of large software systems*, WCRE 2006
- GEVOL - Collberg et al., *A system for graph-based visualization of the evolution of software*, SoftVis 2003
- CVSgrab - Voinea and Telea, *An open framework for CVS repository querying, analysis and visualization*, MSR 2006
- Tesseract - Sarma et al., Tesseract: Interactive visual exploration of socio-technical relationships in software development, ICSE 2009

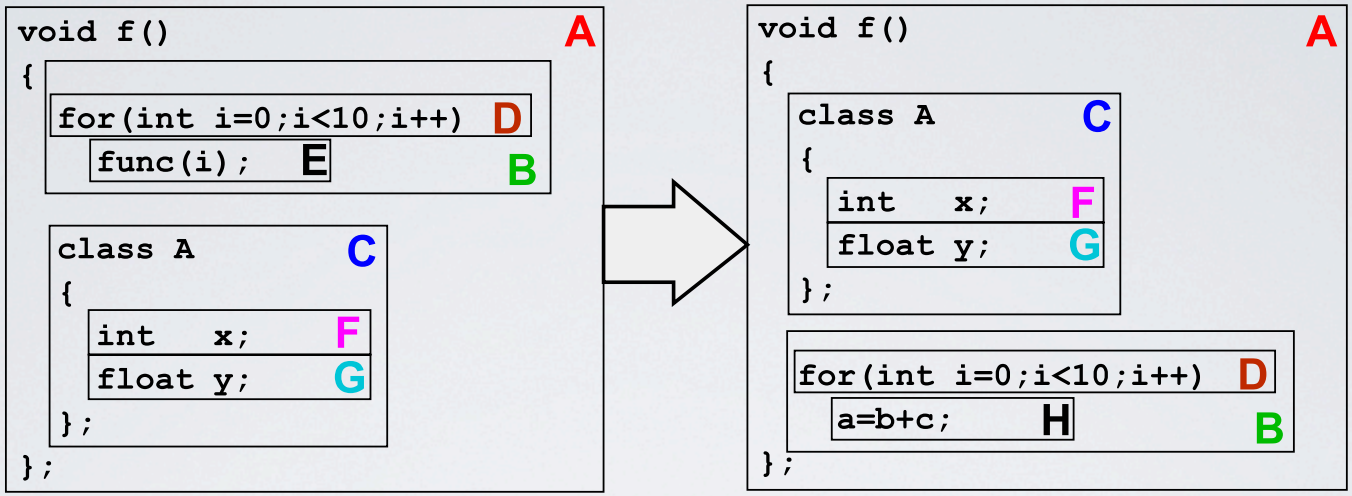
Visual evolution analysis references

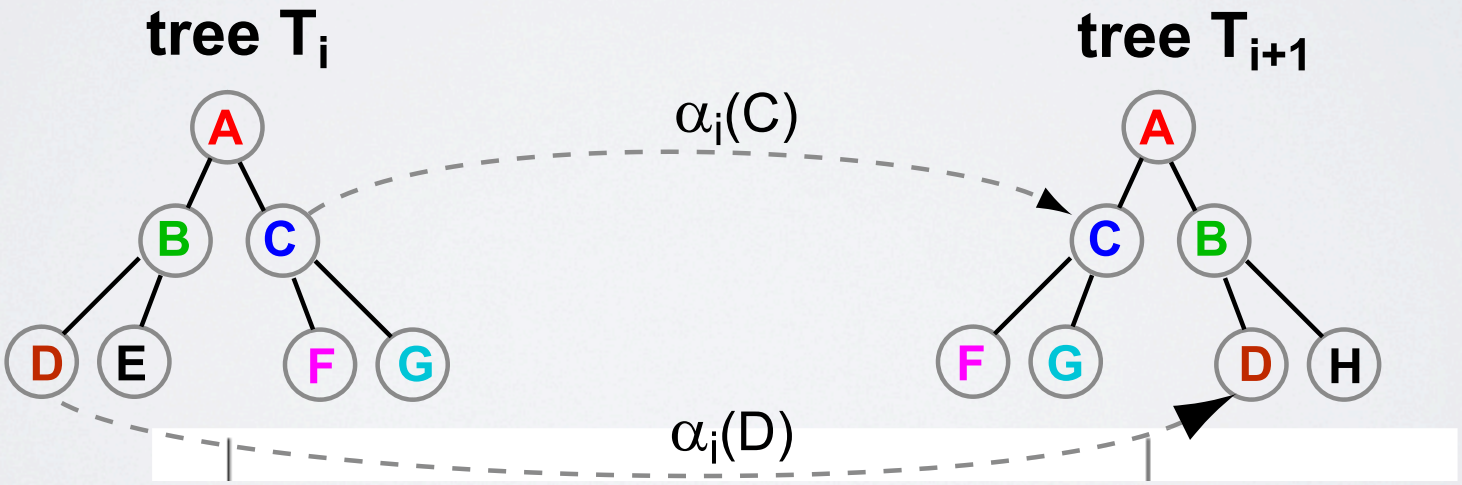
- Intellisense - Holmes and Begel, Deep Intellisense: A tool for rehydrating evaporated information, MSR 2008
- RelVis - Pinzger et al., Visualizing multiple evolution metrics, SoftVis 2005
- YARN - Hindle et al., YARN: Animating software evolution, VISSOFT 2007
- SPO - Lungu et al., *Reverse Engineering Super-Repositories*, WCRE 2007
- Complicity - Neu et al., *Telling Stories about GNOME with Complicity*, VISSOFT 2011
- Ball et al., If Your Version Control System Could Talk, 1997
- Gall et al., Detection of Logical Coupling Based on Product Release History, ICSM 1998
- Gall et al., CVS Release History Data for Detecting Logical Couplings, IWPSE 2003
- Ratzinger et al., Improving evolvability through refactoring, MSR 2005
- Beyer and Hassan, Animated visualization of software history using evolution storyboards, WCRE 2006
- D'Ambros et al., Visualizing Co-Change Information with the Evolution Radar, TSE 2009

Visual evolution analysis summary

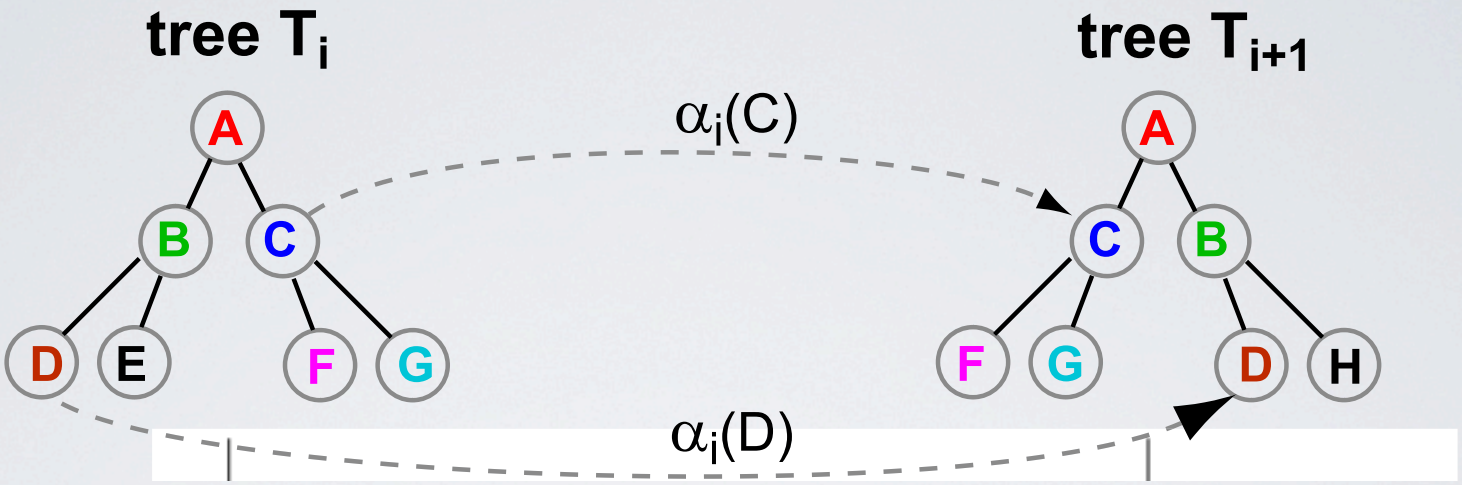
	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows	• SeeSoft	• Augur • CVSscan
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

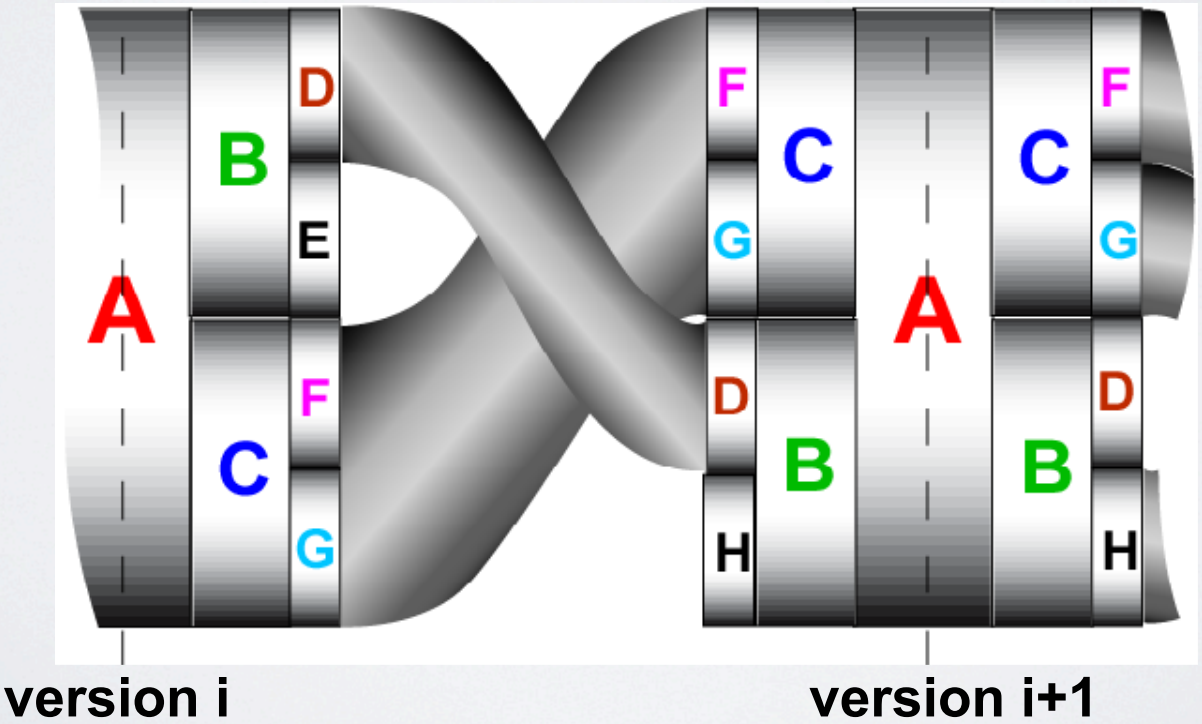
Code Flows principles



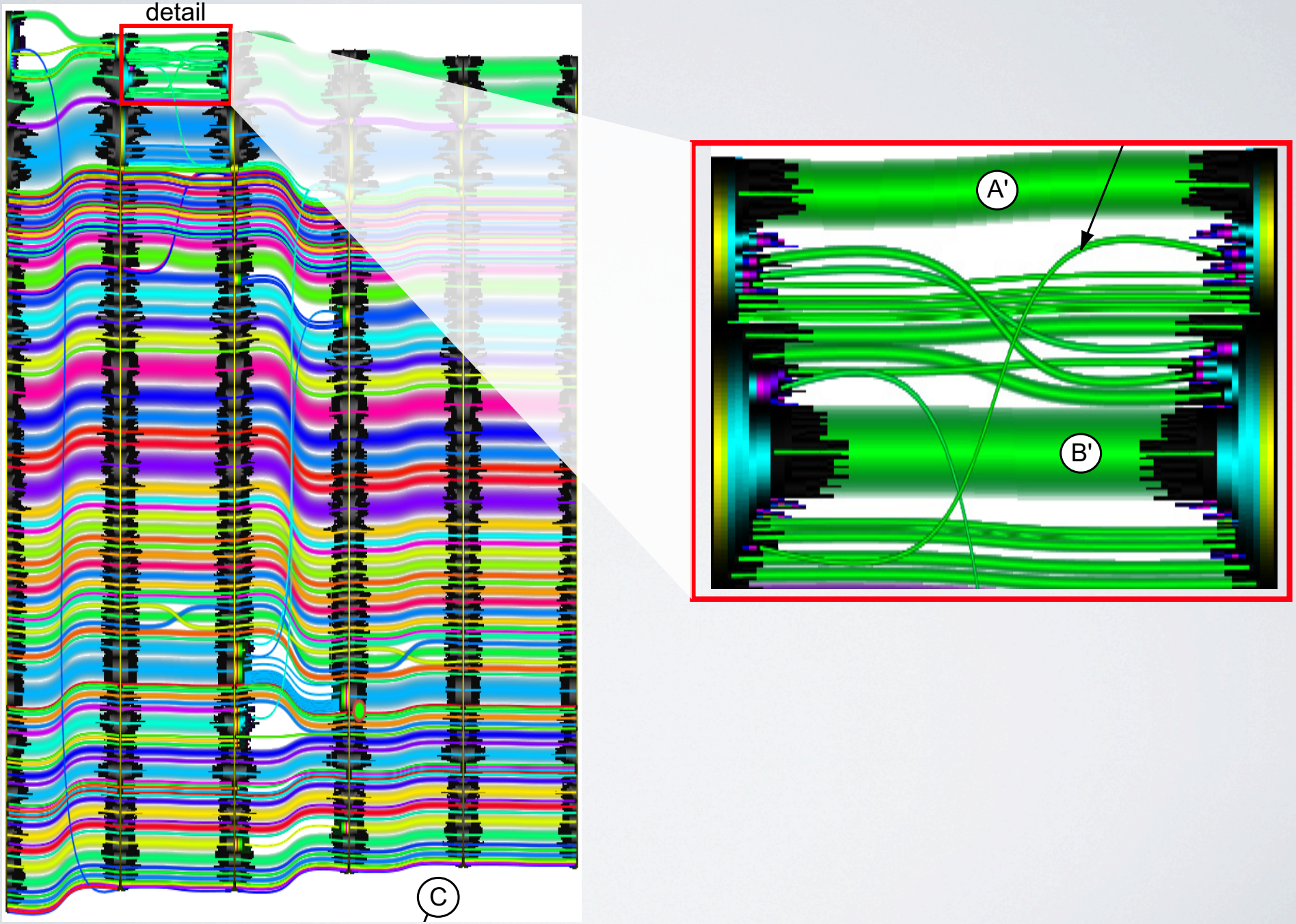


Code Flows principles





Code Flows example



Visual evolution analysis summary

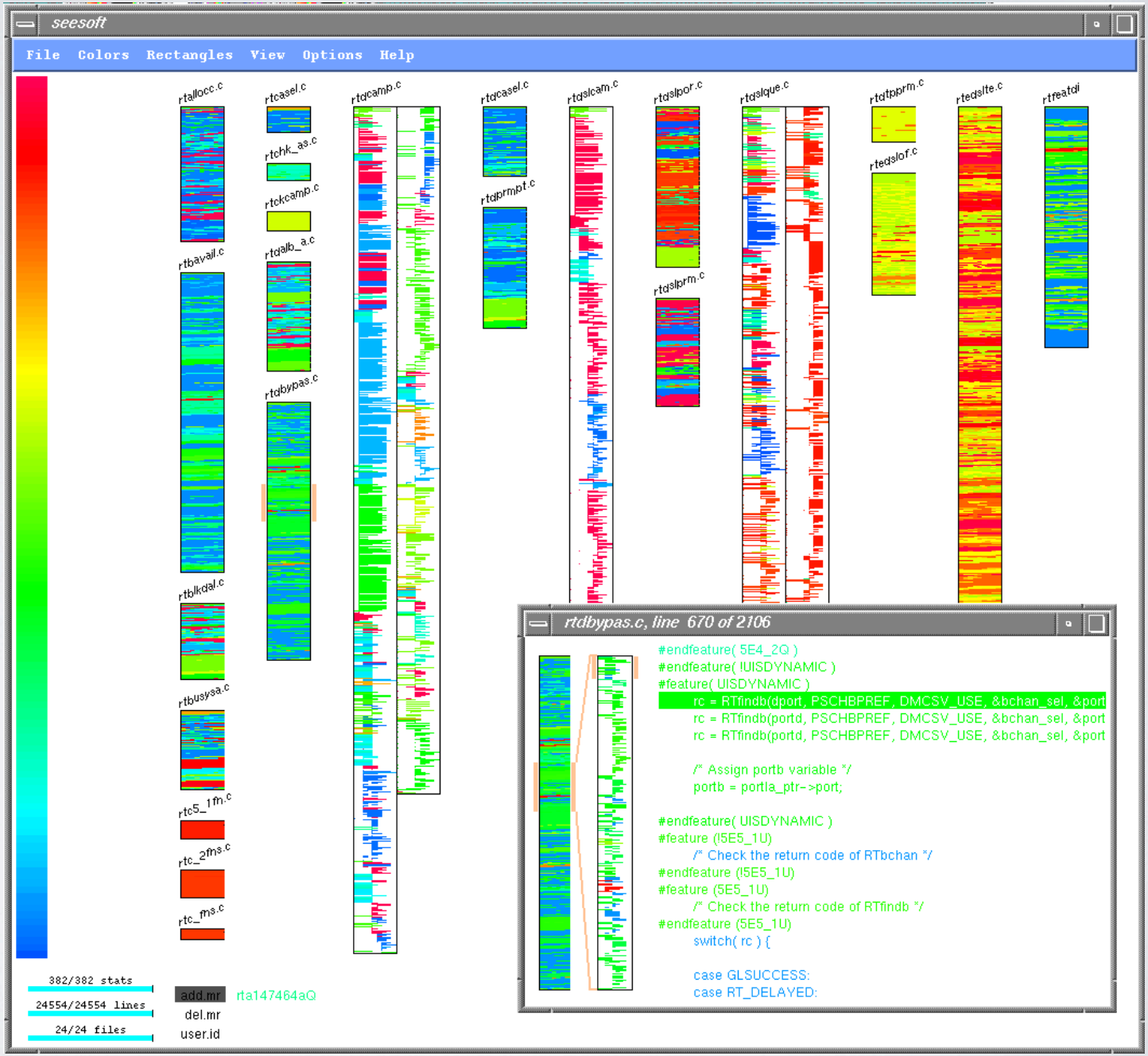
	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows 	• SeeSoft	• Augur • CVSscan
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows 	• SeeSoft	• Augur • CVSscan
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

SeeSoft example - Code age (1996!)

New lines →



Old lines →

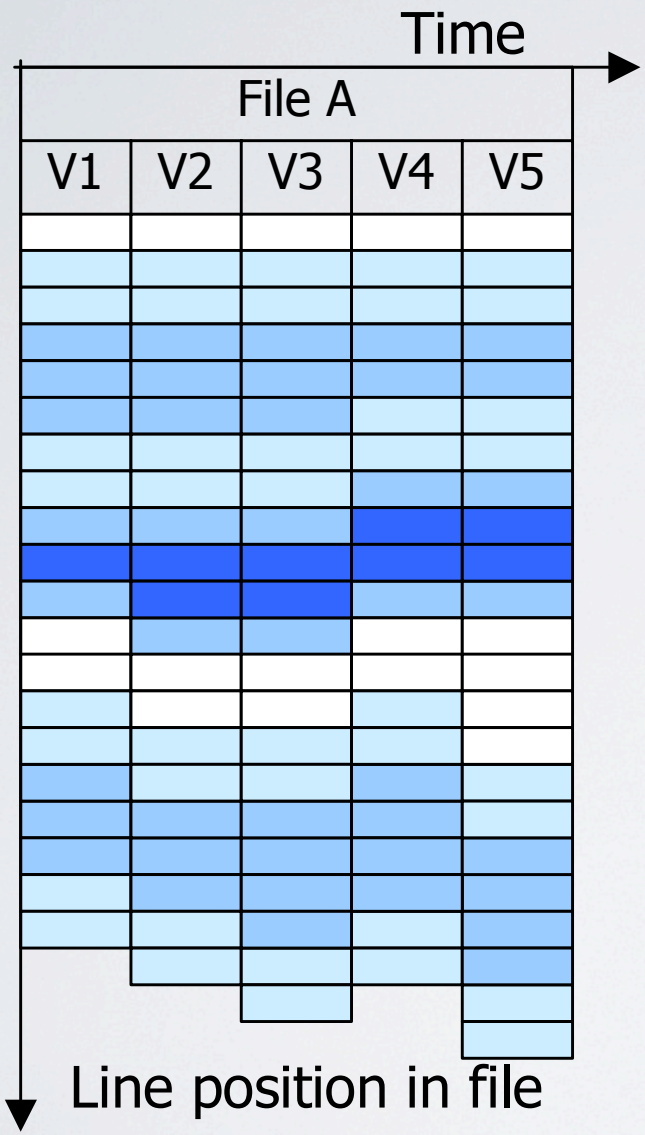
Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

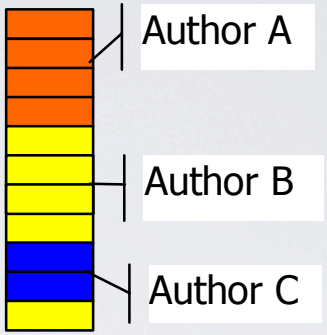
Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

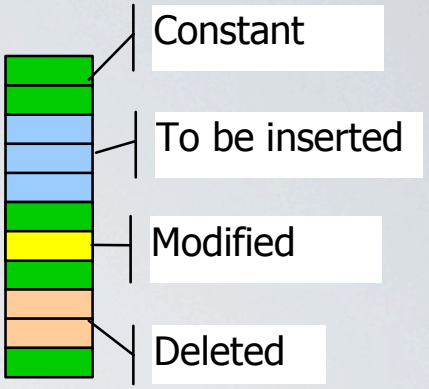
CVSScan principles



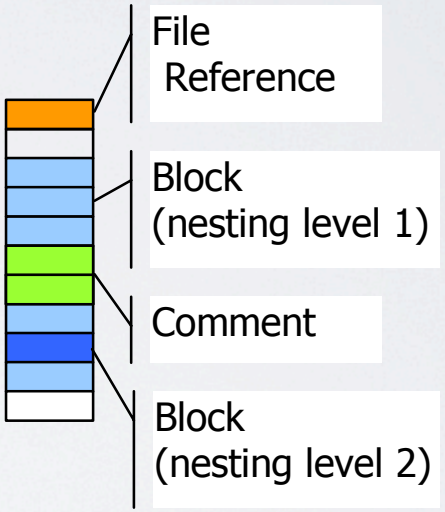
Layout



Author



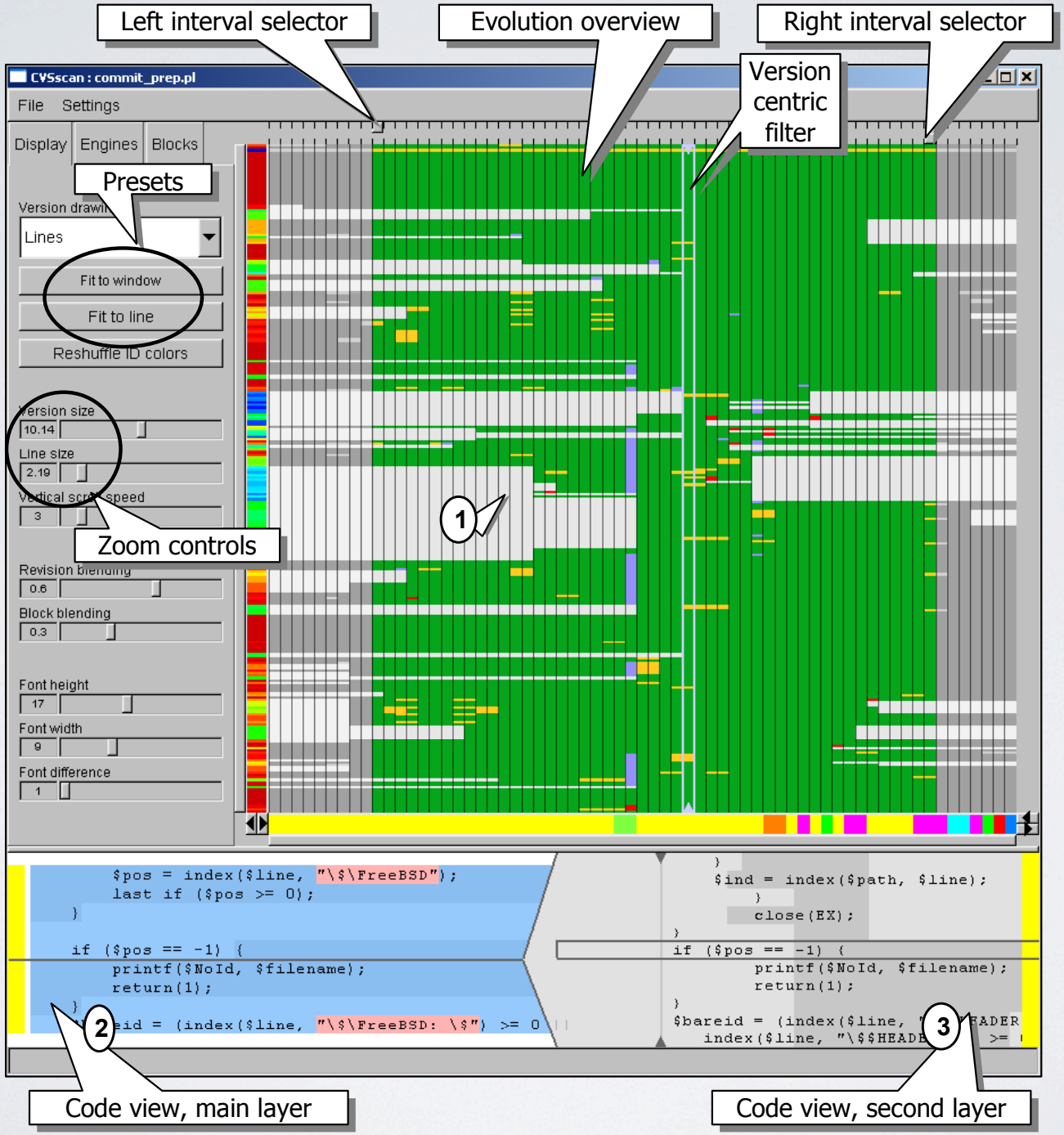
Line status



Construct

Color mappings

CVSScan example



Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

Visual evolution analysis summary

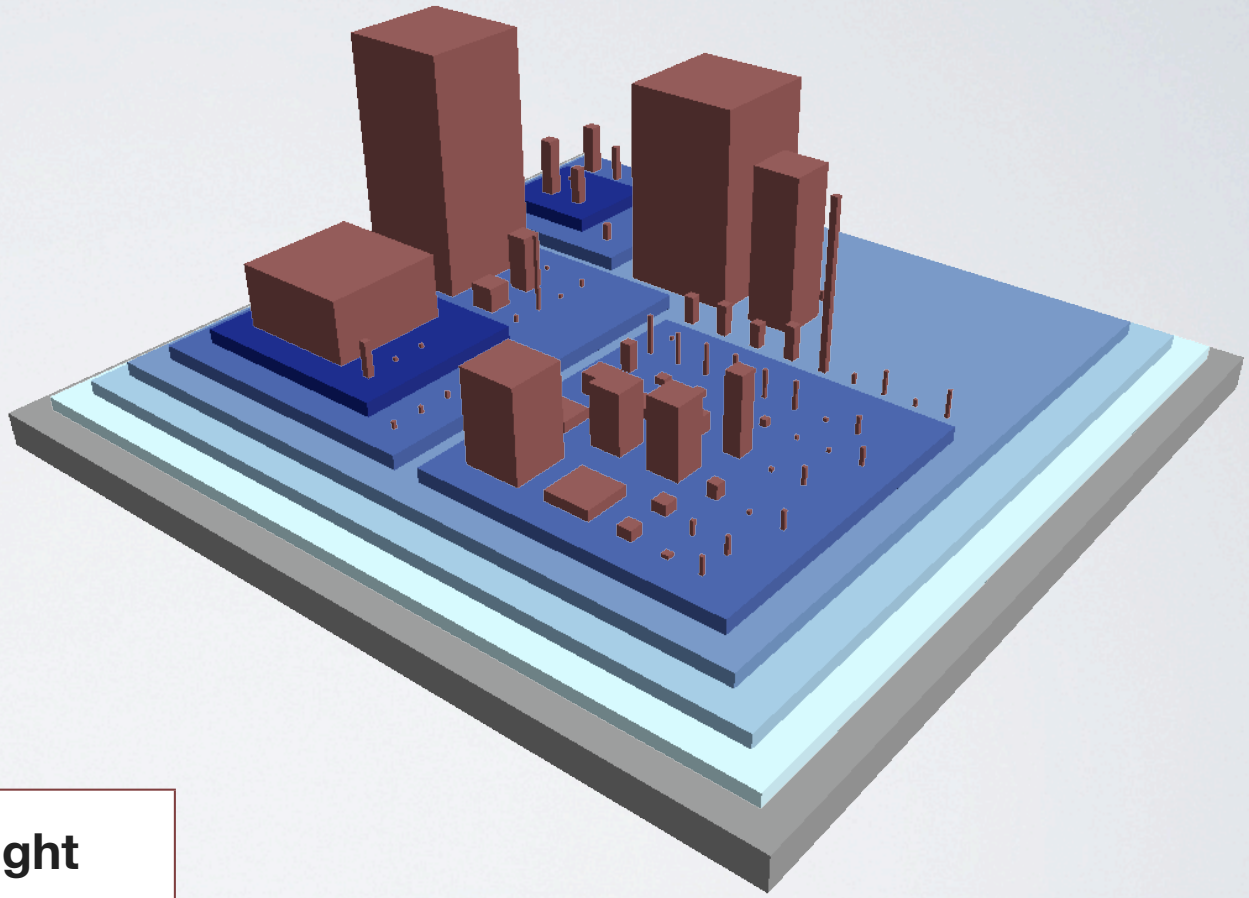
	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

CodeCity - principles

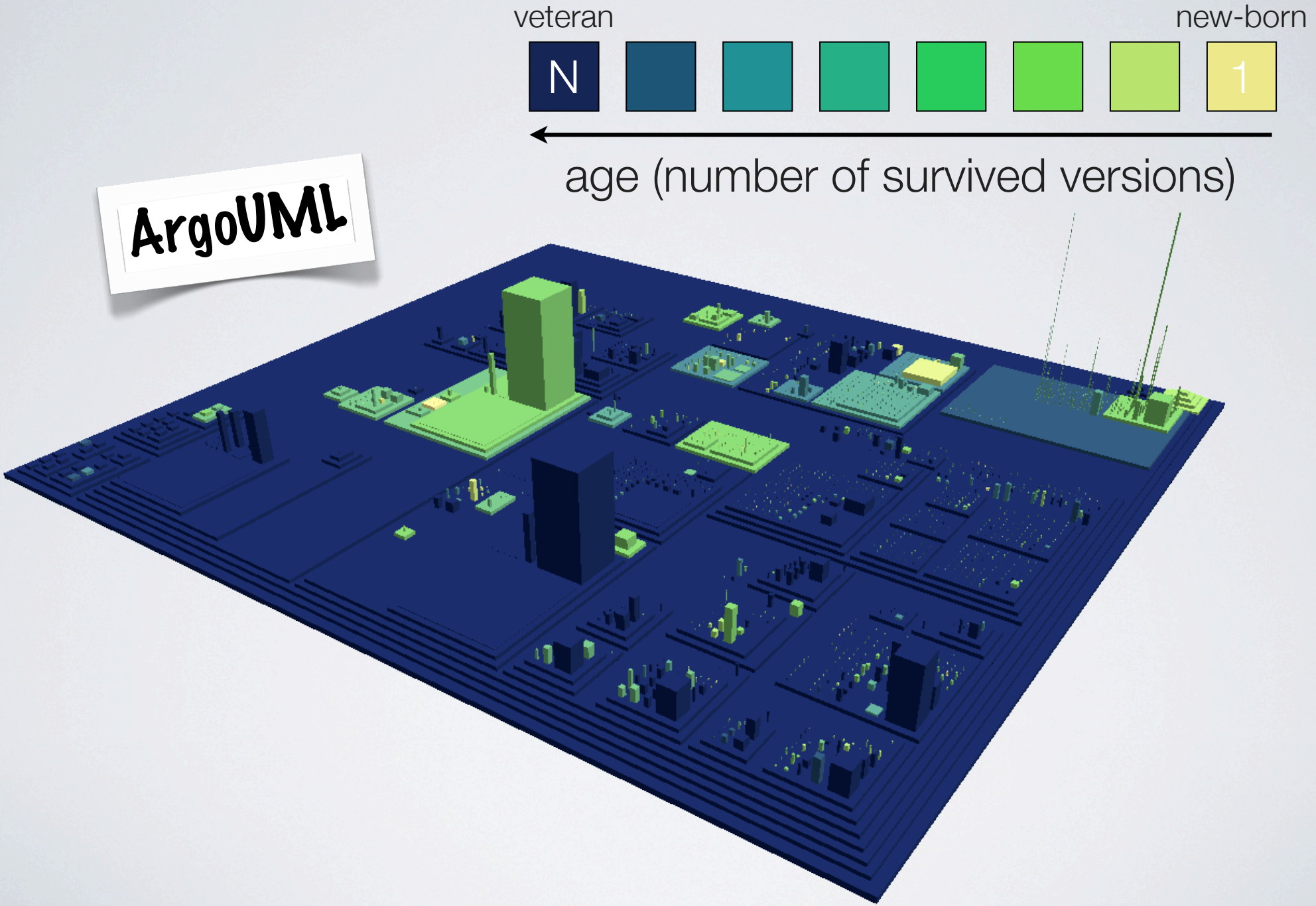
domain mapping	
class	**building**
package	**district**
system	**city**

nesting level	**color**

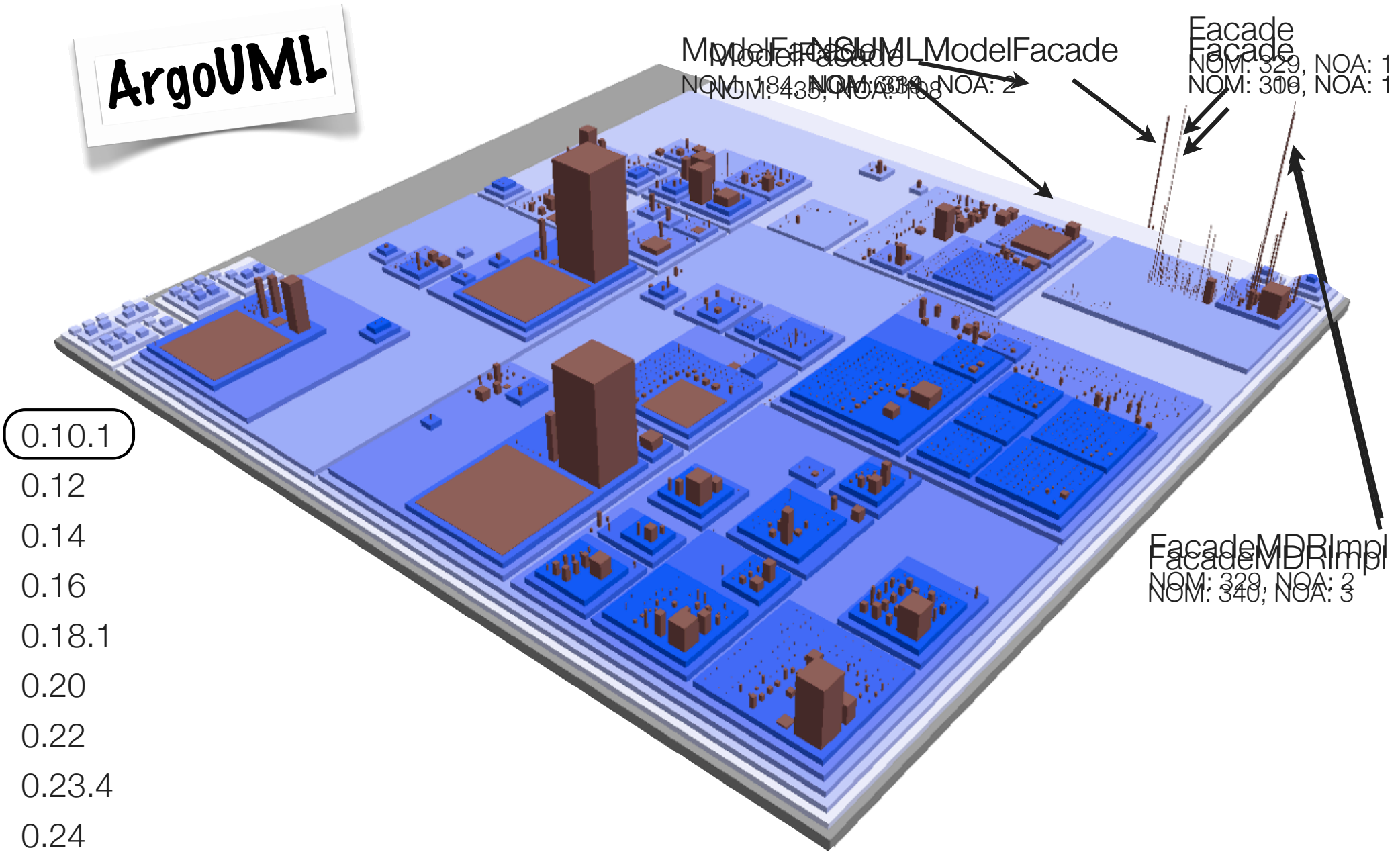
number of methods (NOM)	**height**
number of attributes (NOA)	**base size**



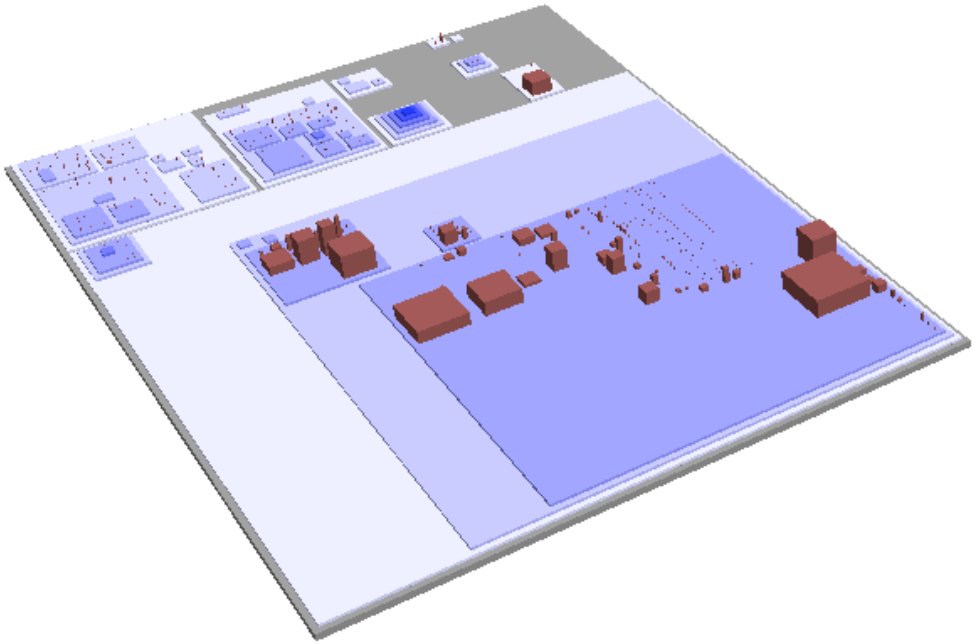
CodeCity - age map



CodeCity - time travel



CodeCity - The story of JMol

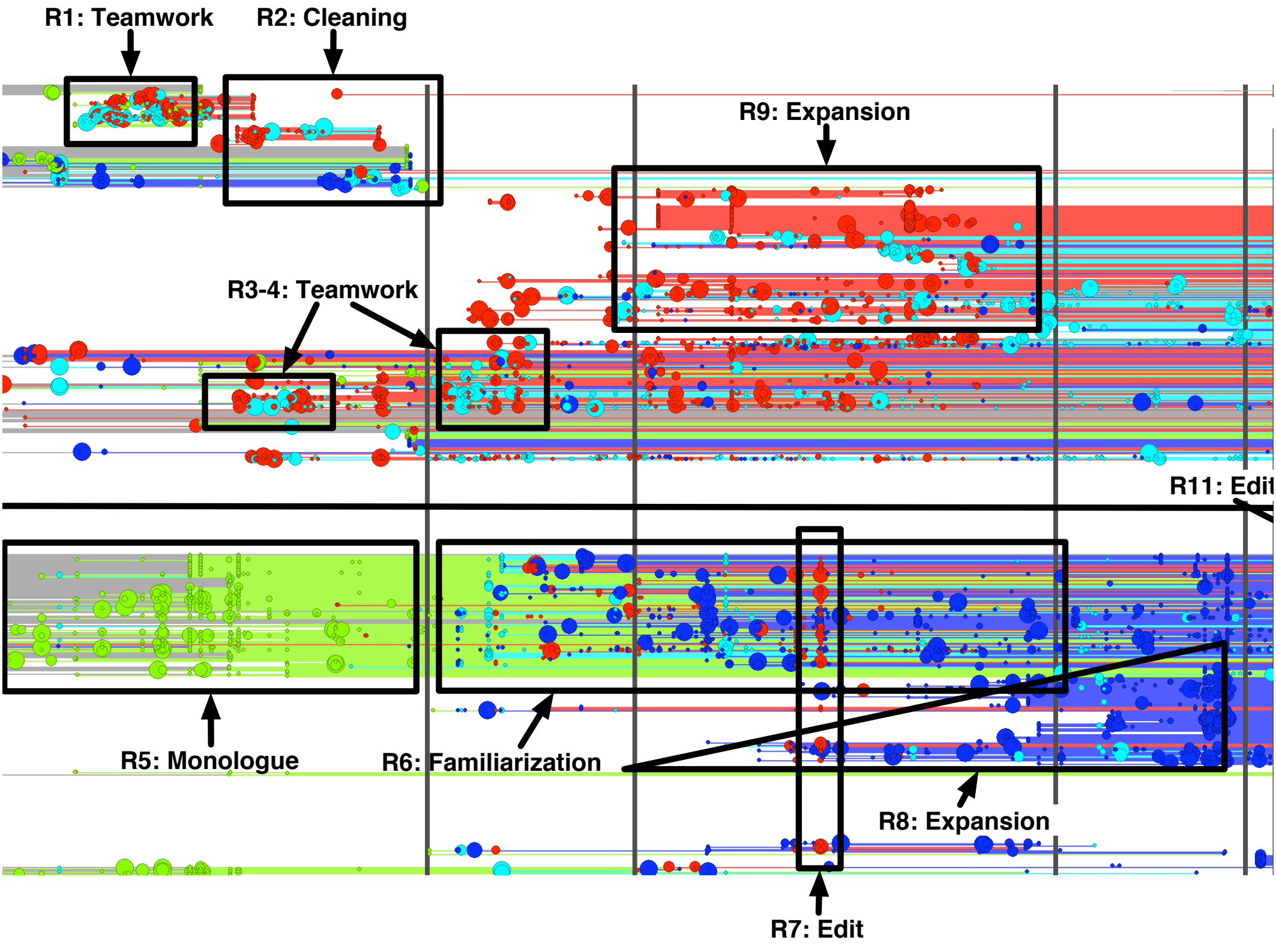


Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

Visual evolution analysis summary

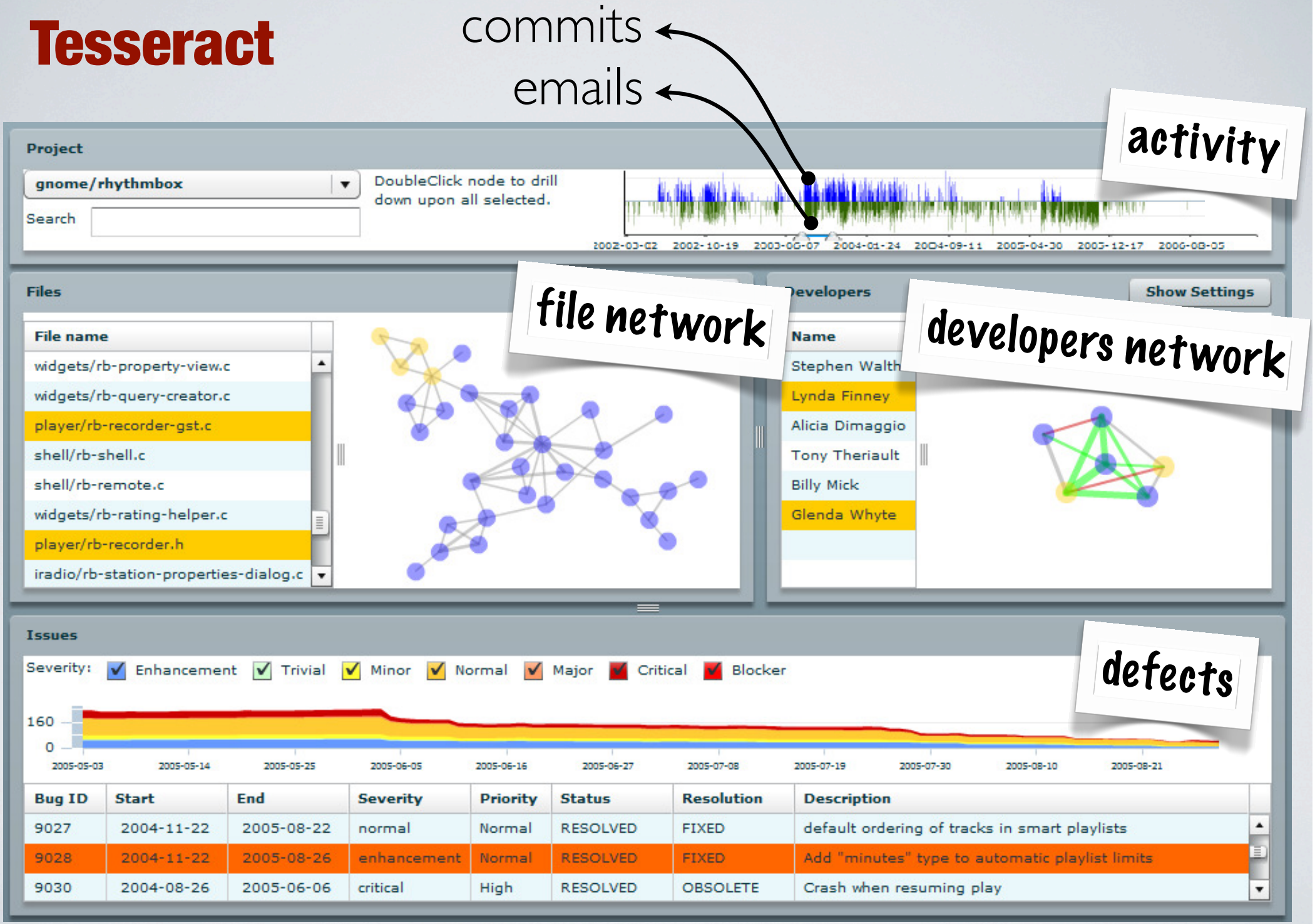
	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn


R1: Teamwork
R2: Cleaning
R9: Expansion
R3-4: Teamwork
R11: Edit
R5: Monologue
R6: Familiarization
R8: Expansion
R7: Edit

Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows	• SeeSoft	• Augur • CVSscan
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

Tesseract


commits
emails
activity
Project
gnome/rhythmbox
Search
DoubleClick node to drill down upon all selected.
2002-03-02
2002-10-19
2003-06-07
2004-01-24
2004-09-11
2005-04-30
2005-12-17
Files
file network
File name
widgets/rb-property-view.c
widgets/rb-query-creator.c
player/rb-recorder-gst.c
shell/rb-shell.c
shell/rb-remote.c
widgets/rb-rating-helper.c
player/rb-recorder.h
iradio/rb-station-properties-dialog.c
Developers
Show Settings
developers network
Name
Stephen Walth
Lynda Finney
Alicia Dimaggio
Tony Theriault
Billy Mick
Glenda Whyte
Issues
defects
Severity:
Enhancement
Trivial
Minor
Normal
Major
Critical
Blocker
160
0
2005-05-03
2005-05-14
2005-05-25
2005-06-05
2005-06-16
2005-06-27
2005-07-08
2005-07-19
2005-07-30
2005-08-10
2005-08-21
Bug ID
Start
End
Severity
Priority
Status
Resolution
Description
9027
2004-11-22
2005-08-22
normal
Normal
RESOLVED
FIXED
default ordering of tracks in smart playlists
9028
2004-11-22
2005-08-26
enhancement
Normal
RESOLVED
FIXED
Add "minutes" type to automatic playlist limits
9030
2004-08-26
2005-06-06
critical
High
RESOLVED
OBSOLETE
Crash when resuming play

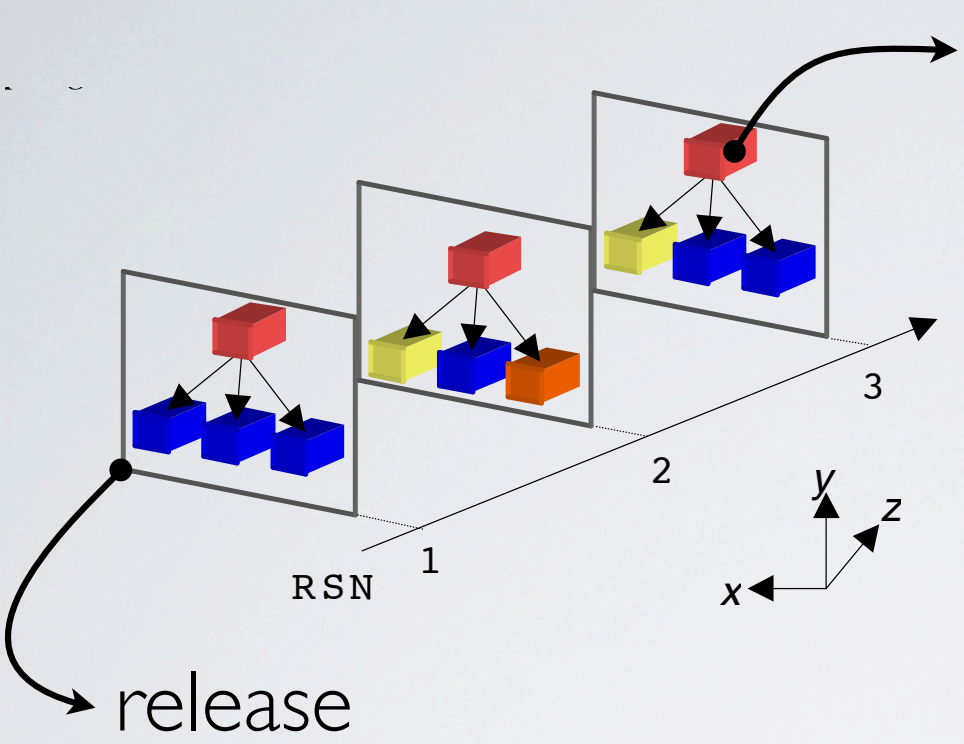
Visual evolution analysis summary

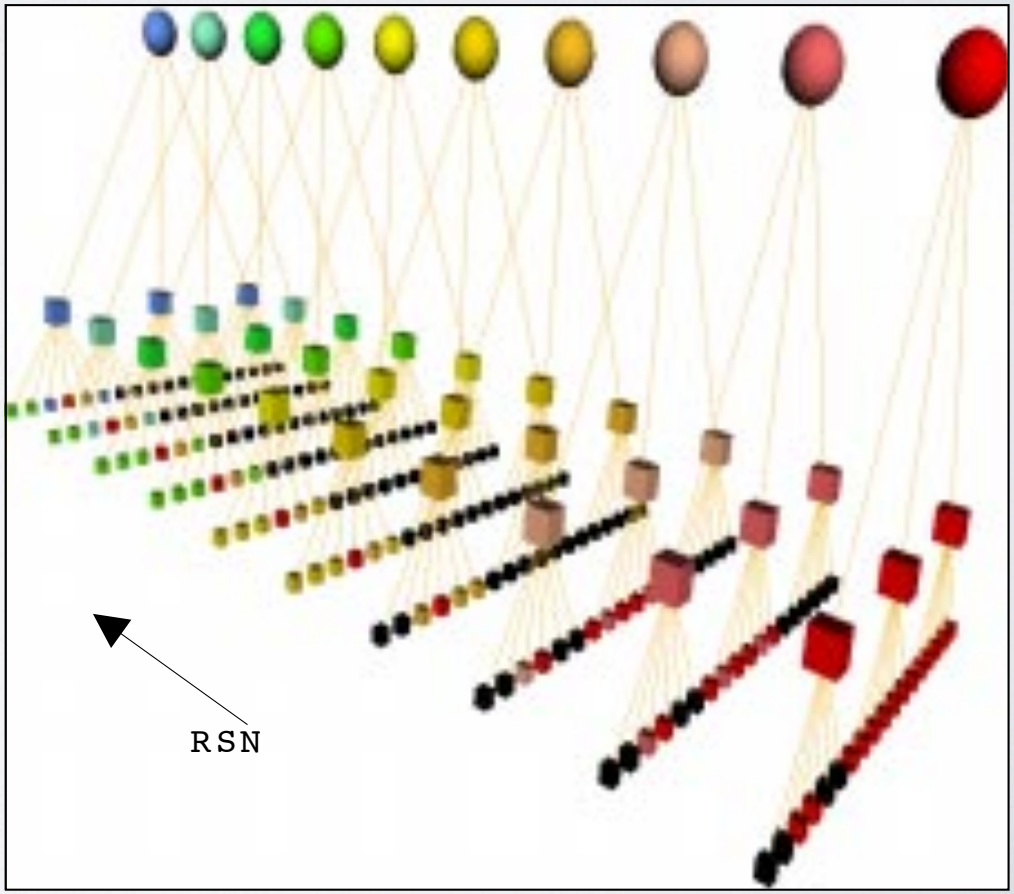
	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract ✓ • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract ✓ • Deep Intellisense
Architecture	• 3DSoftVis • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

3DSoftVis





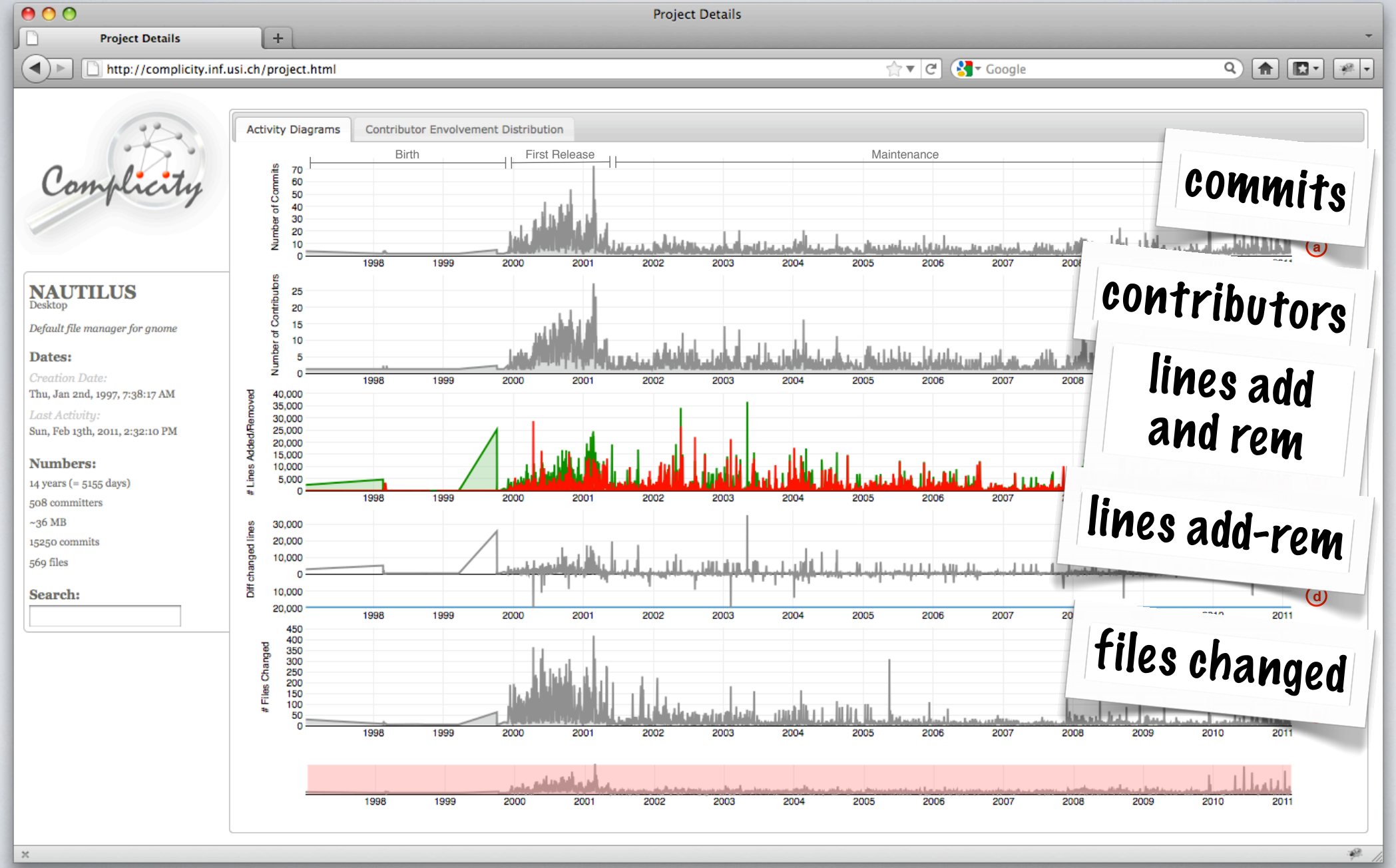
Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract ✓ • Deep Intellisense
Architecture	• 3DSoftVis ✓ • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

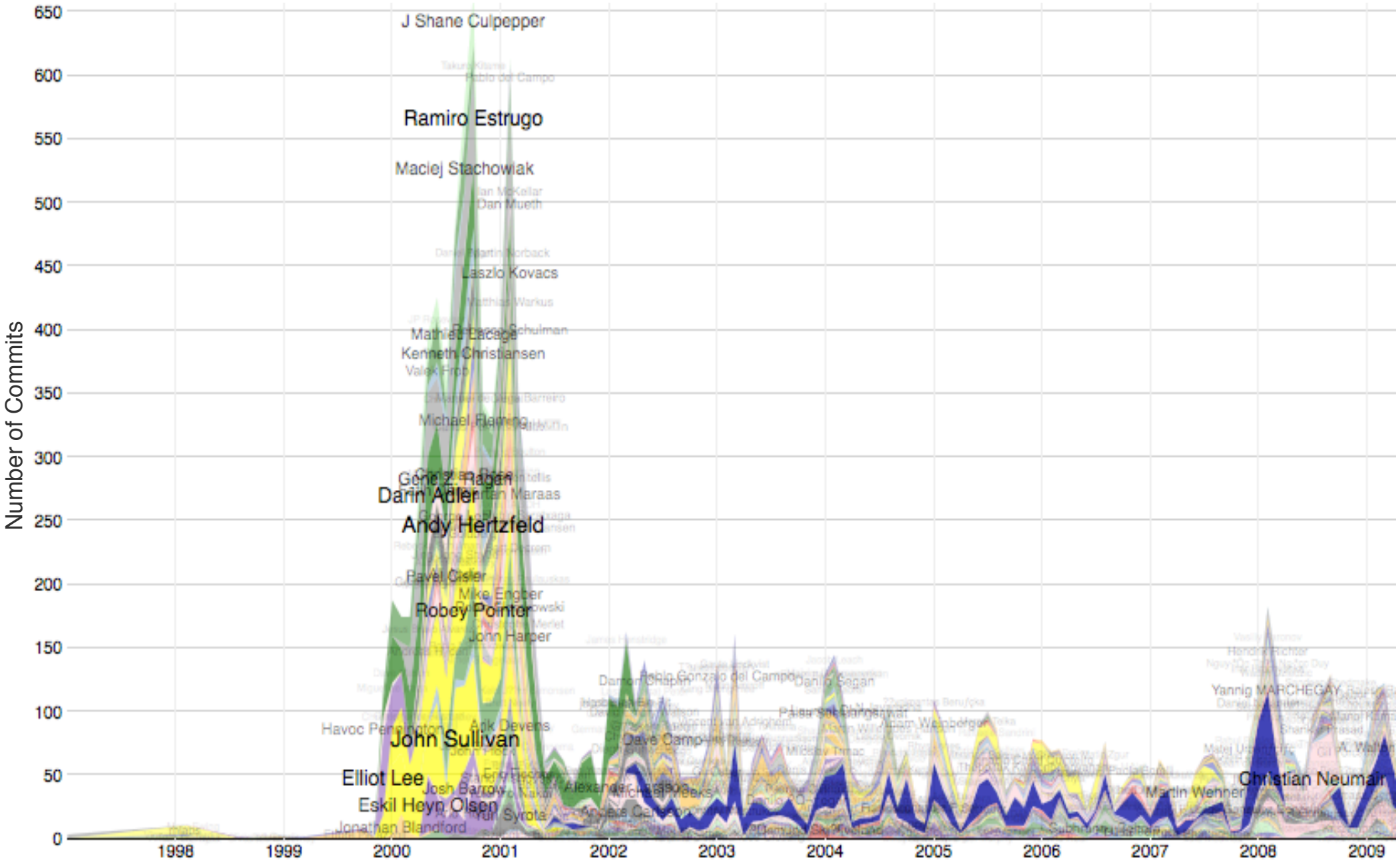
Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract ✓ • Deep Intellisense
Architecture	• 3DSoftVis ✓ • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity	• RelVis • Yarn

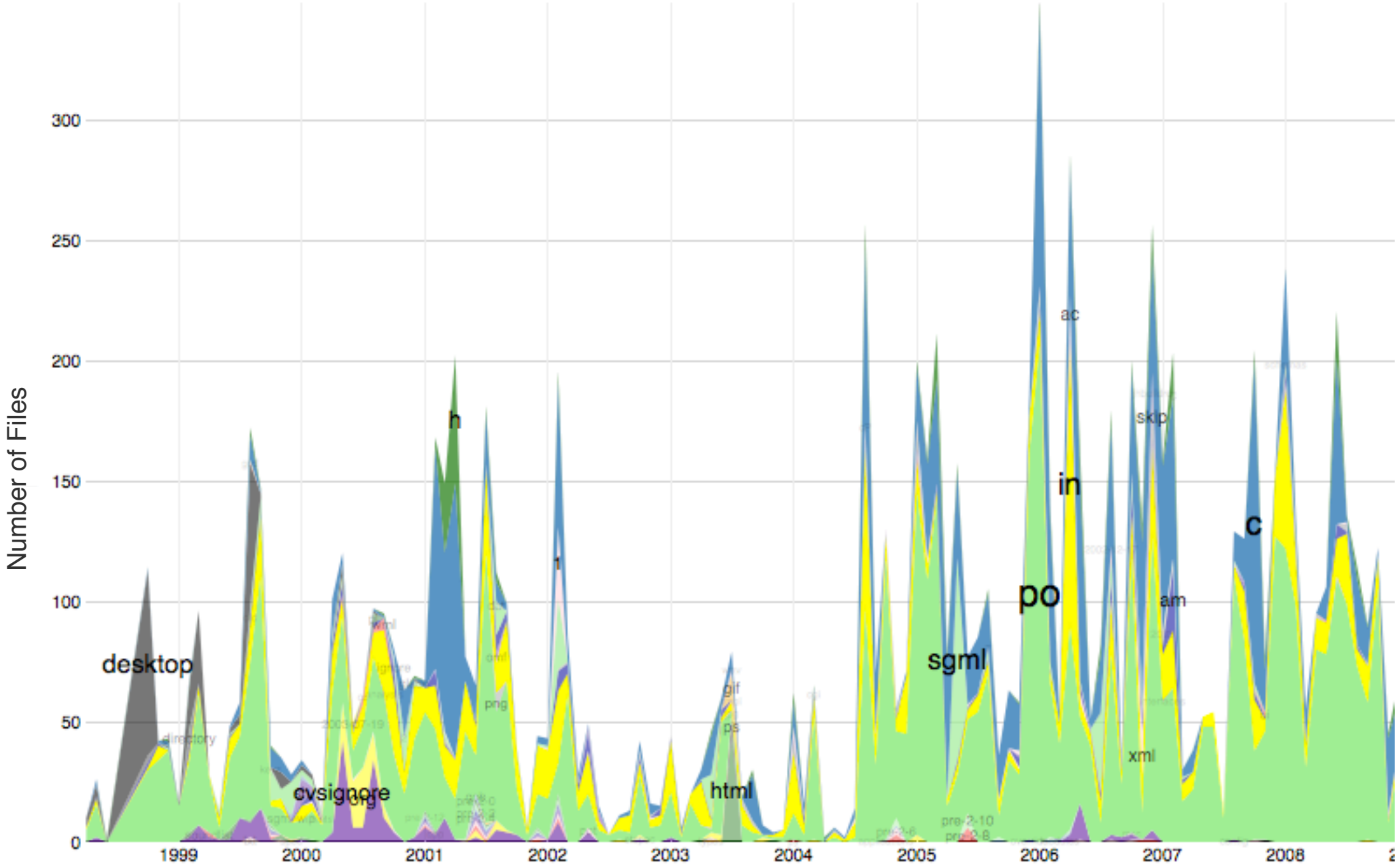
Complicity - Visualizing software ecosystem



Complicity - contributors involvement



Complicity - expertise view



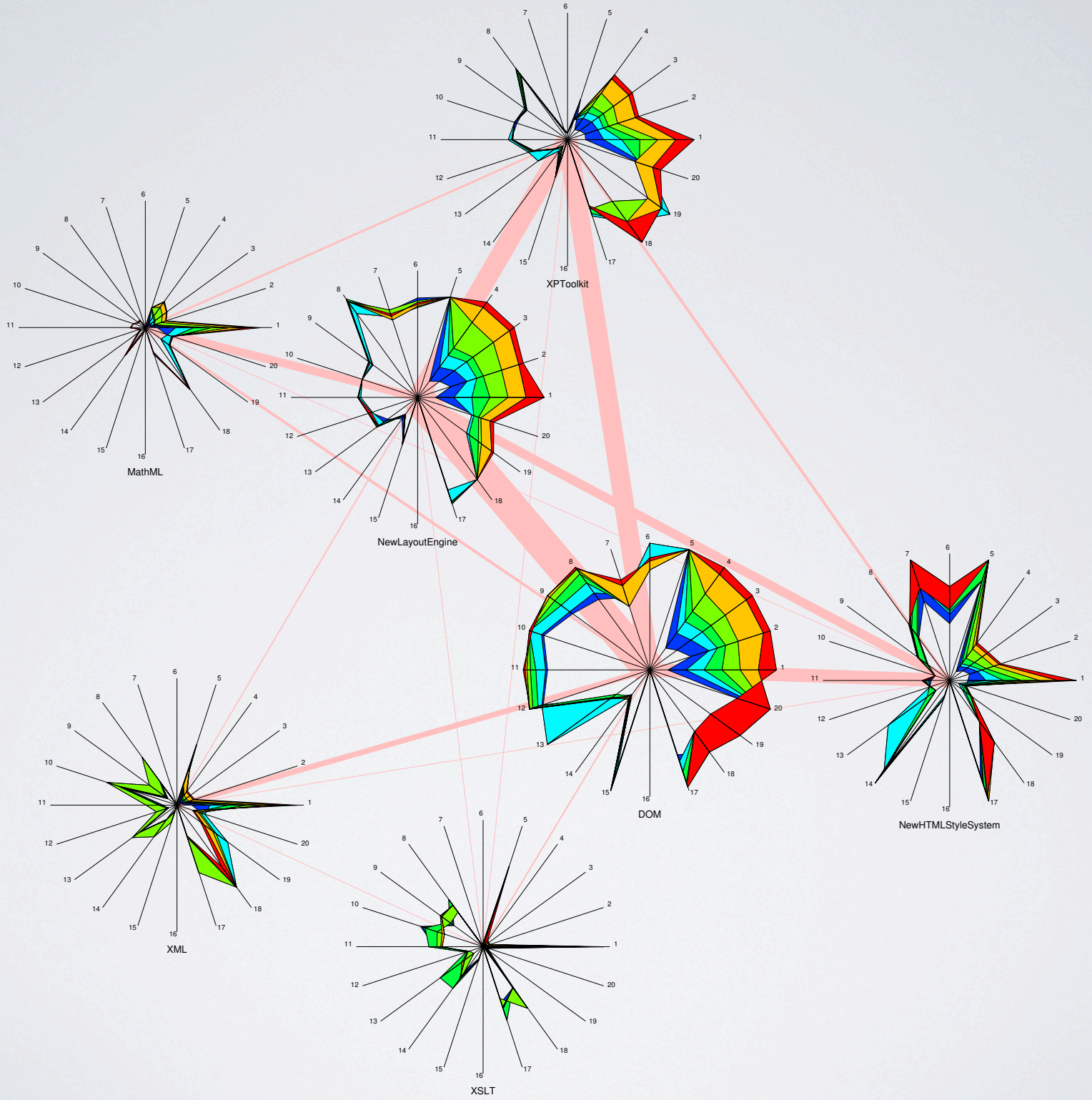
Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract ✓ • Deep Intellisense
Architecture	• 3DSoftVis ✓ • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity ✓	• RelVis • Yarn

Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract ✓ • Deep Intellisense
Architecture	• 3DSoftVis ✓ • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity ✓	• RelVis ✓ • Yarn

RelVis


XPToolkit
MathML
NewLayoutEngine
DOM
NewHTMLStyleSystem
XML
XSLT

Visual evolution analysis summary

	Multiple snapshots	Versioning system log files	Integrated data sources
Code	• Code Flows ✓	• SeeSoft ✓	• Augur • CVSscan ✓
Design	• CodeCrawler • The Evolution Matrix • BEAGLE • CodeCity ✓	• Revision Towers • Rysselberghe and Demeyer 2004 • Evolution Spectrographs • Chronia ✓ • Evolens • Change coupling (previous slides)	• BugCrawler • EvoGraph • Gevol • CVSgrab • Tesseract ✓ • Deep Intellisense
Architecture	• 3DSoftVis ✓ • Jazayeri 2002	• Change coupling (previous slides) • SPO • Complicity ✓	• RelVis ✓ • Yarn

Visual Evolution Analysis

- Goal: Visualize evolutionary data to:
  - understanding a software system's evolution
  - support decision making
  - detect design problems (e.g. architecture decay)

Defect prediction

Change analysis and prediction

Empirical studies

Expertise & bug triaging

Visual evolution analysis

Human factors

Human factors

- Goal: Extract, measure and analyze communication and organizational information to
  - improve previous MSR techniques
  - support software maintenance and development
  - perform empirical studies

“Design and programming are human activities; forget that and all is lost”

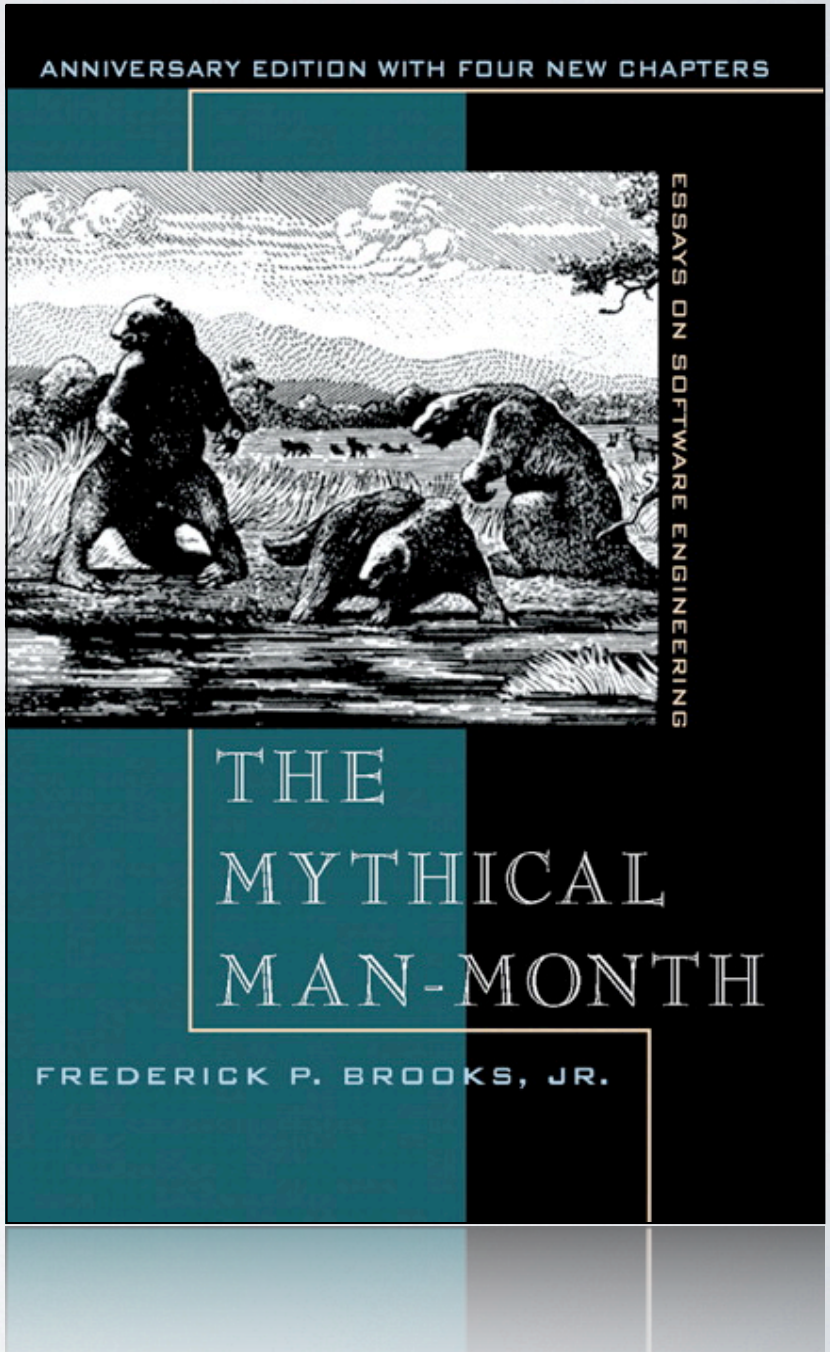
— Bjarne Stroustrop

Impact of the communication structure

> *"Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure"*

> *"A software system whose structure closely matches its organization's communication structure works "better" than a subsystem whose structure differs from its organization's communication structure"*

— Melvin Conway, 1968



Empirical investigation of Conway's law

- Nagappan et al. proposed a metric scheme to quantify organizational complexity, measuring issues such as:
  - organizational distance of the developers
  - the amount of multi-tasking developers are doing across organizations
  - the amount of change to a component within the context of that organization
- They then investigated (on Windows Vista) whether this metrics are correlated with software quality

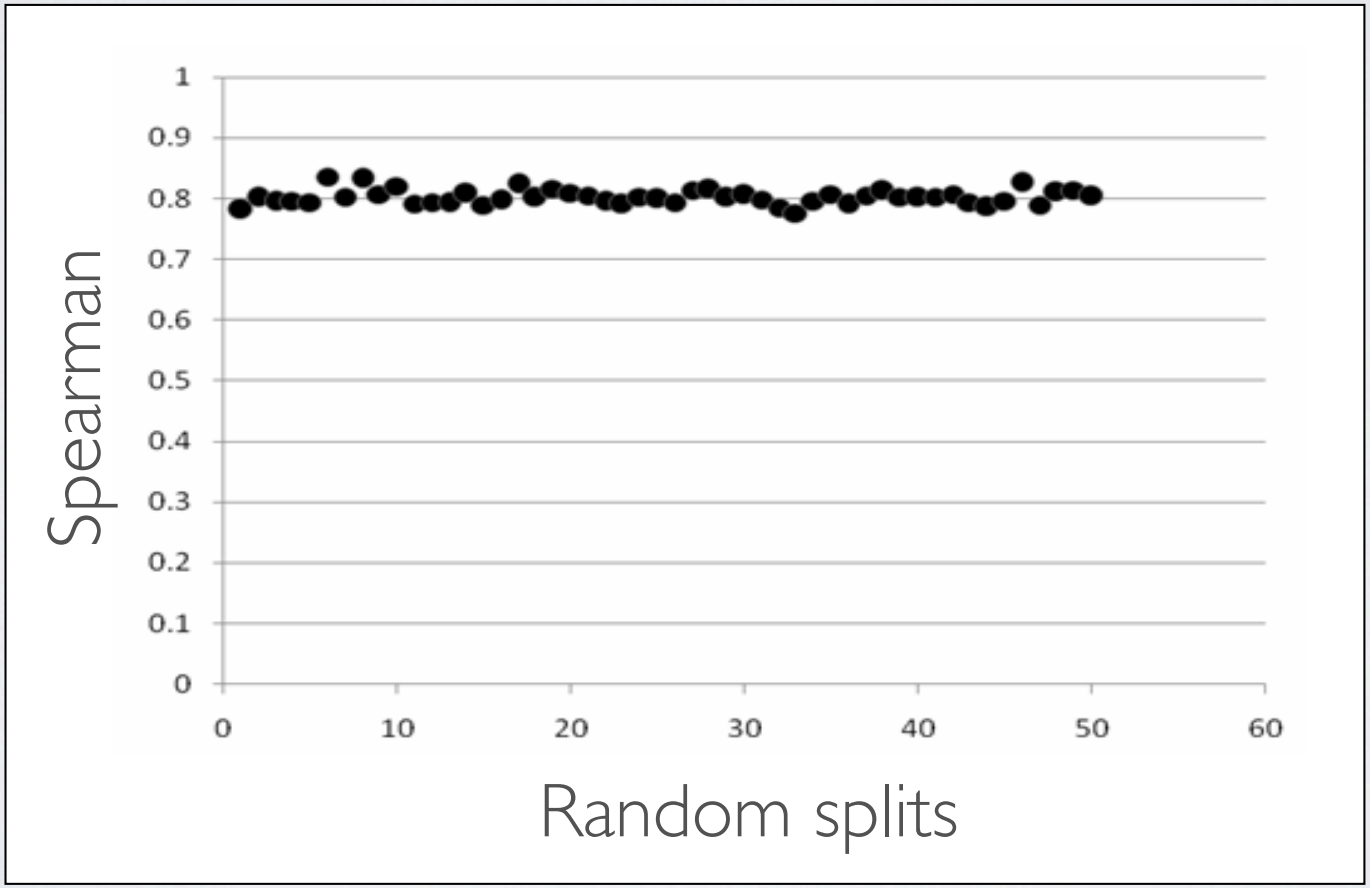
Nagappan et al., *The influence of organizational structure on software quality: An empirical case study*, ICSE 2008

Organizational metrics

- Number of (ex-)engineers: how many engineers "touched" a binary and are still employed by the company (have left the company)
- Edit frequency: how many times an artifact was edited
- Depth of master ownership: the level of ownership of the binary depending on the number of edits done
- Level of Organizational Code Ownership: the percent of edits from the organization that contains the binary
- Organization Intersection Factor: the number of different organizations that contribute greater than 10% of edits
- ...

Using organizational metrics to predict defects

- Predicting post-release defects on Windows Vista at the binary level



Using communication metrics to predict defects

- Predicting post-release defects on Windows Vista at the binary level

Model	Precision	Recall
Code Churn	78.6%	79.9%
Code Complexity	79.3%	66.0%
Dependencies	74.4%	69.9%
Code Coverage	83.8%	54.4%
Pre-Release Bugs	73.8%	62.9%
Organizational Structure	**86.2%**	**84.0%**

Using communication metrics to predict defects

- Predicting post-release defects on Windows Vista at the binary level

Model	Precision	Recall
Code Churn	78.6%	79.9%
Code Complexity	79.3%	66.0%
Code Coverage	83.8%	54.4%
Pre-Release Bugs	73.8%	62.9%
Organizational Structure	**86.2%**	**84.0%**

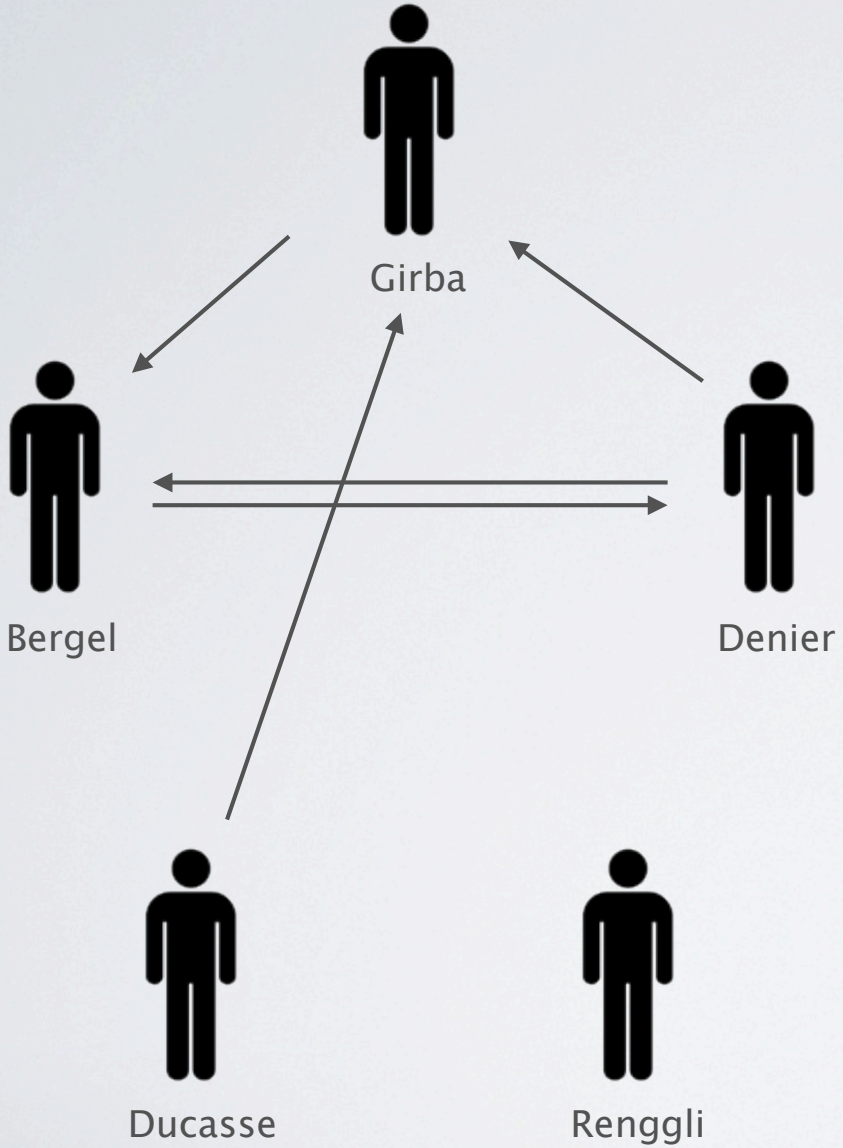
Organizational metrics are far better predictors of defects than source code metrics

Learning from OSS

- Traditional commercial software projects have a carefully structured and hierarchical team organization
- In contrast, OSS projects have no pre-designed organizational structure, but they are large and successful
  - Do they have some latent structure of their own?
  - Are there dynamic, self- organizing subgroups that spontaneously form and evolve?
- Bird et al. answered these questions for 5 successful OSS (httpd, Ant, Python, Perl, PostgreSQL)

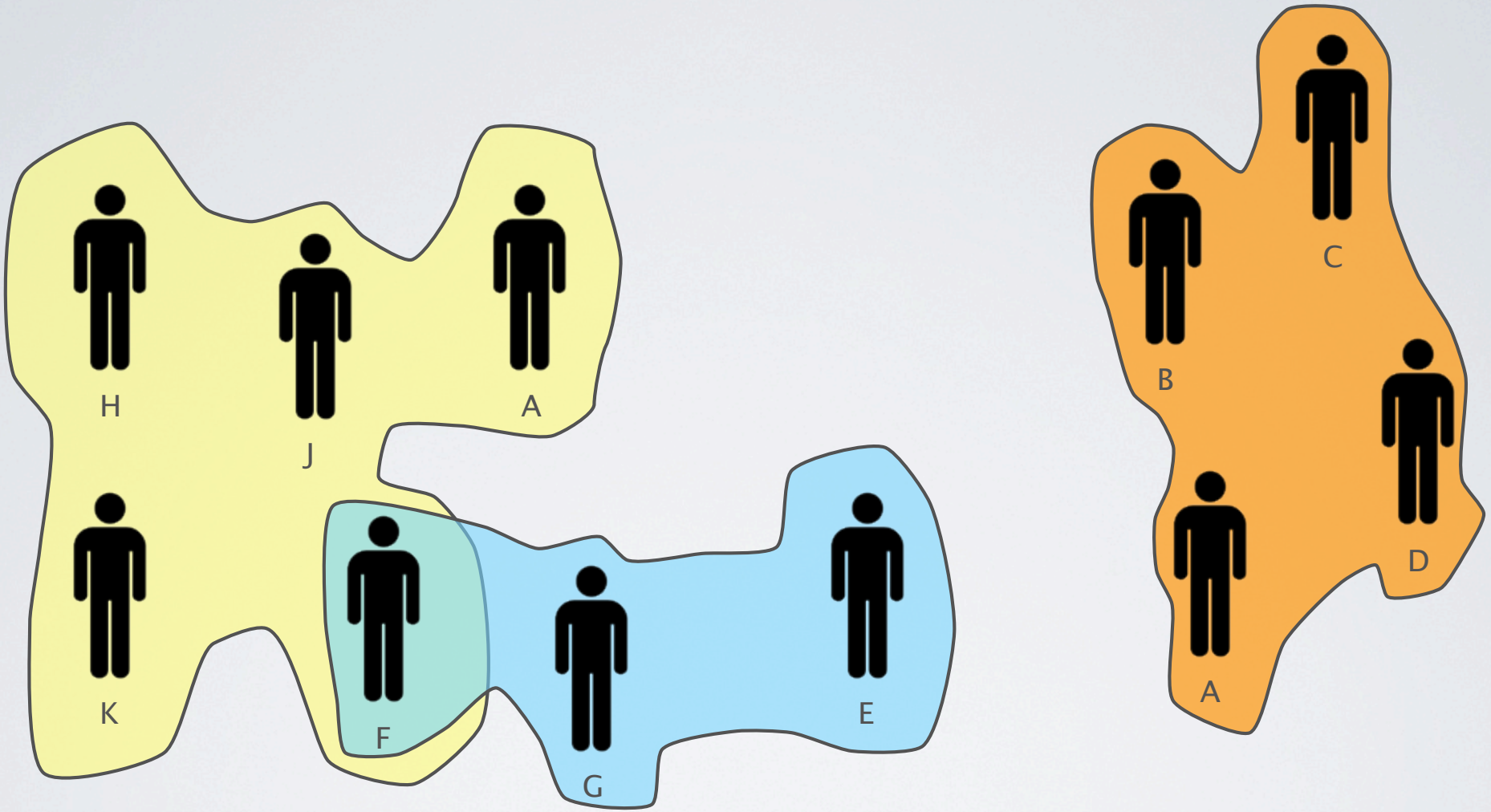
Bird et al., *Latent Social Structure in Open Source Projects*, FSE 2008

Extracting the social structure



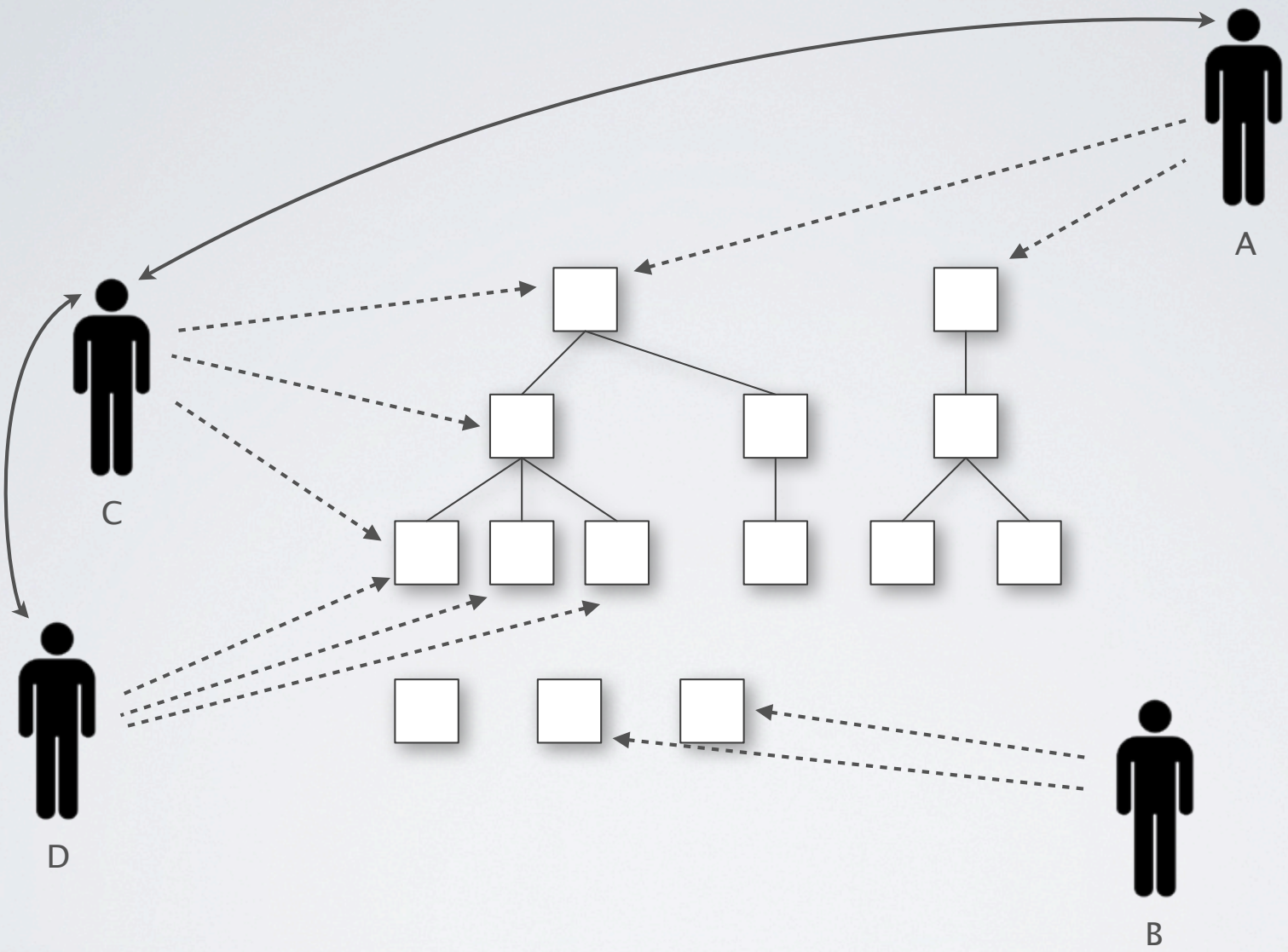
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Extracting the social structure



The social structure is well modularized

Extracting the technical structure



The technical structure is well modularized

Conclusion

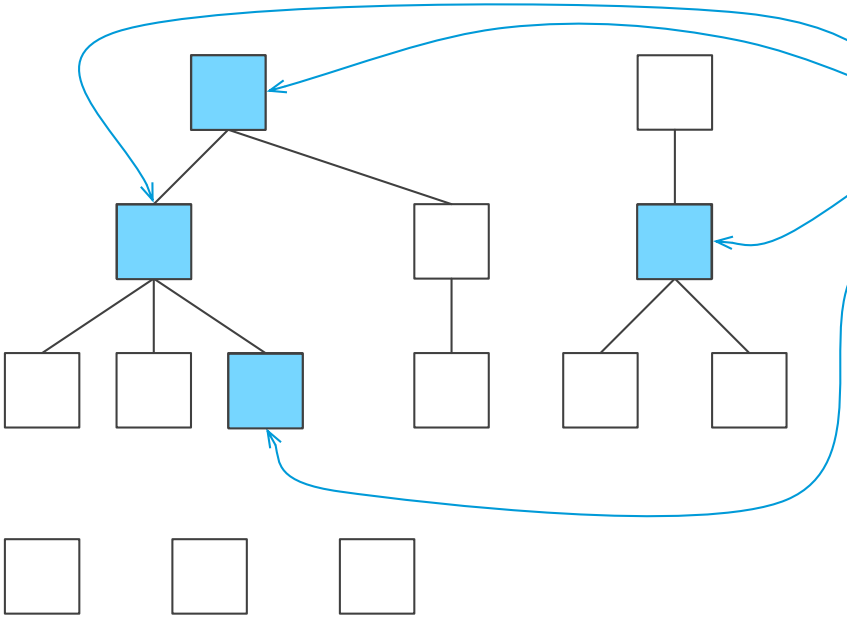
- OSS projects are well modularized and the technical structure reflects the social one
  - Evidence of strong community structure existed in all projects
  - The structure was more modular when discussion focused directly on source code artifacts
  - The division of the project into subcommunities is also representative of the collaboration behavior

Are popular classes more defect prone?

- Assumption: software artifacts that are frequently discusses in development mailing list are more defect prone
- Bacchelli et al. investigated this assumption on 4 OSS (Equinox, Lucene, Maven, JackRabbit)

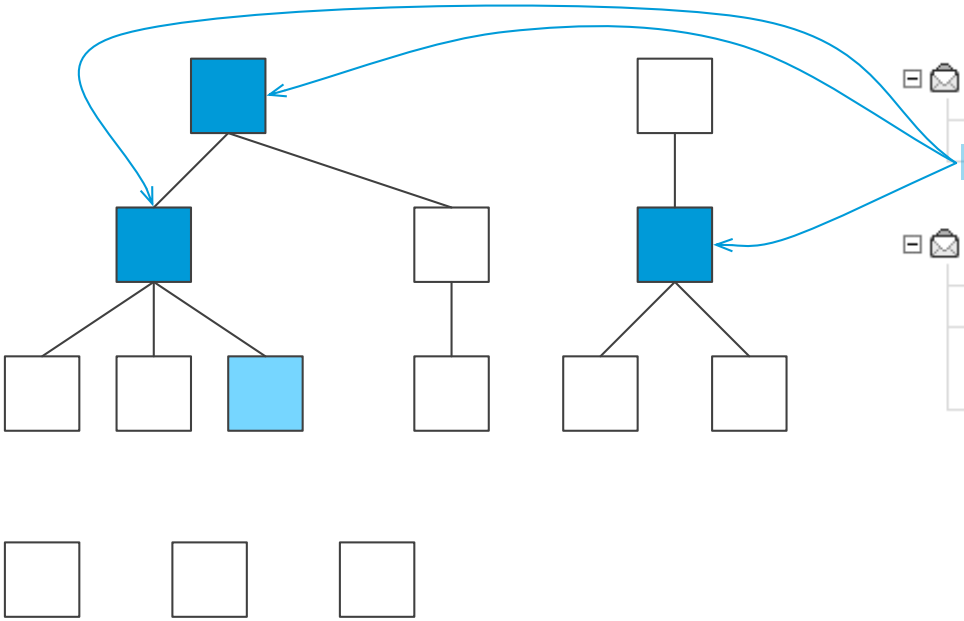
Bacchelli et al., *Are popular classes more defect prone?* FASE 2010

Measuring the popularity of classes



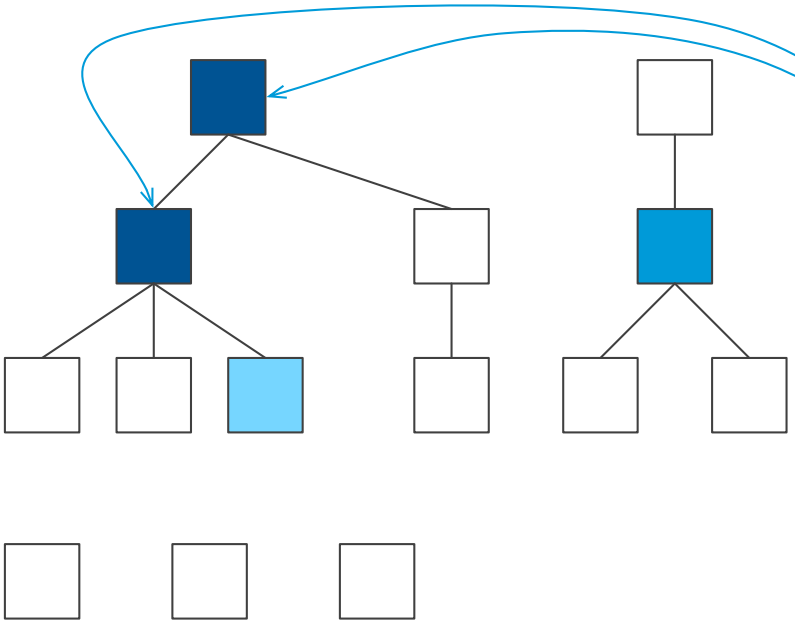
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Measuring the popularity of classes



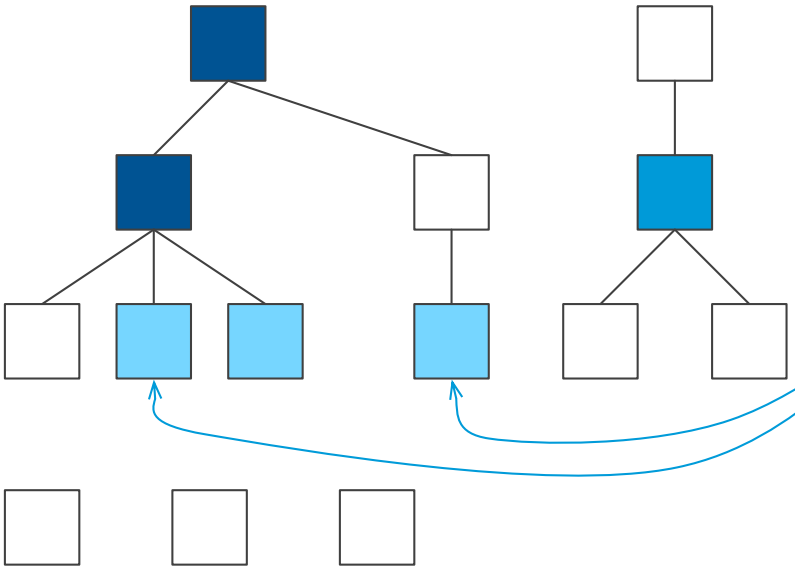
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Measuring the popularity of classes



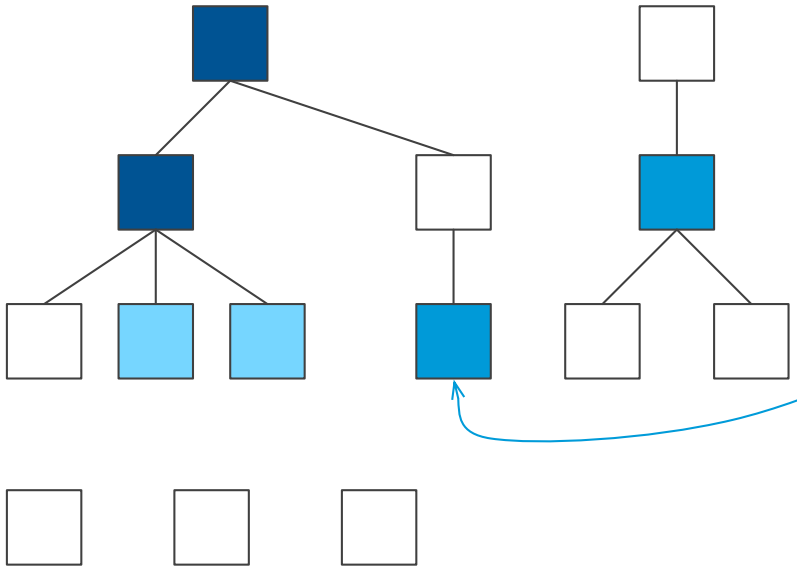
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Measuring the popularity of classes



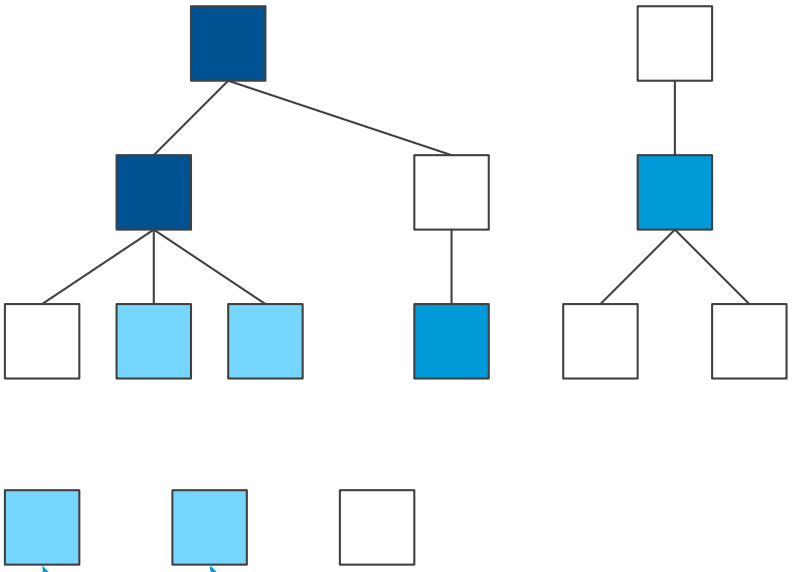
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Measuring the popularity of classes



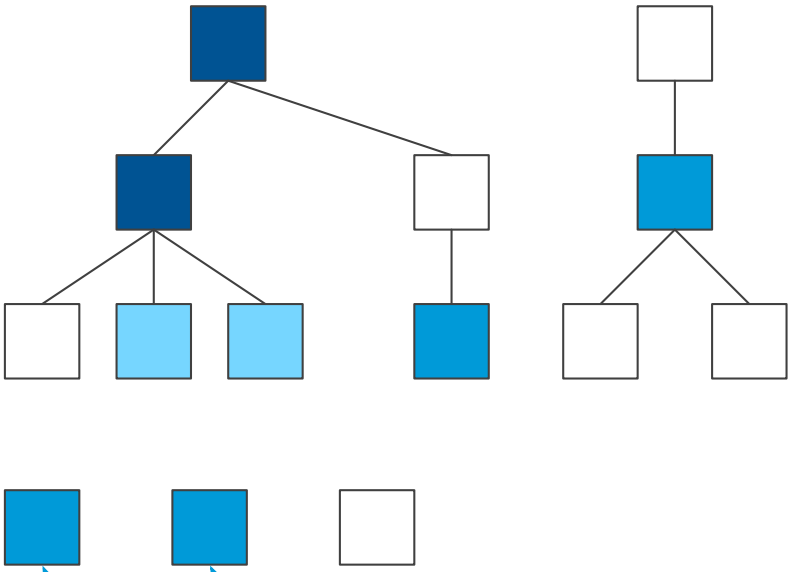
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Measuring the popularity of classes



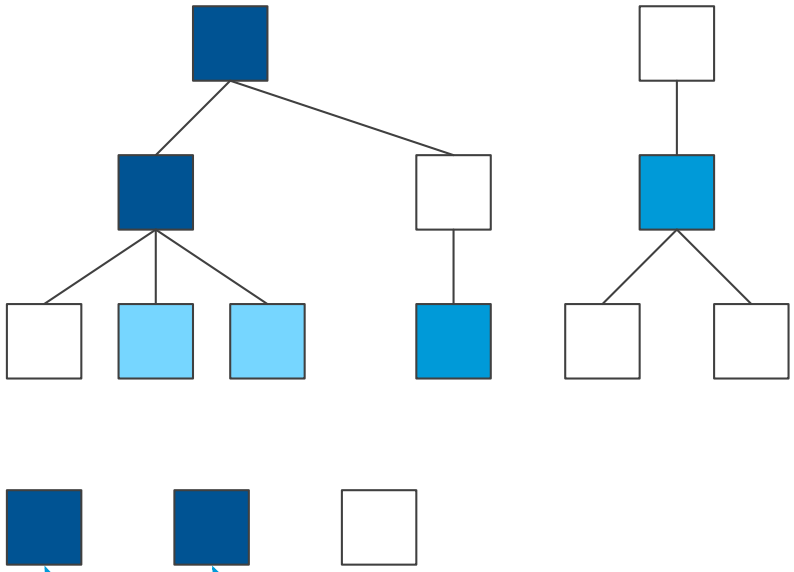
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Measuring the popularity of classes



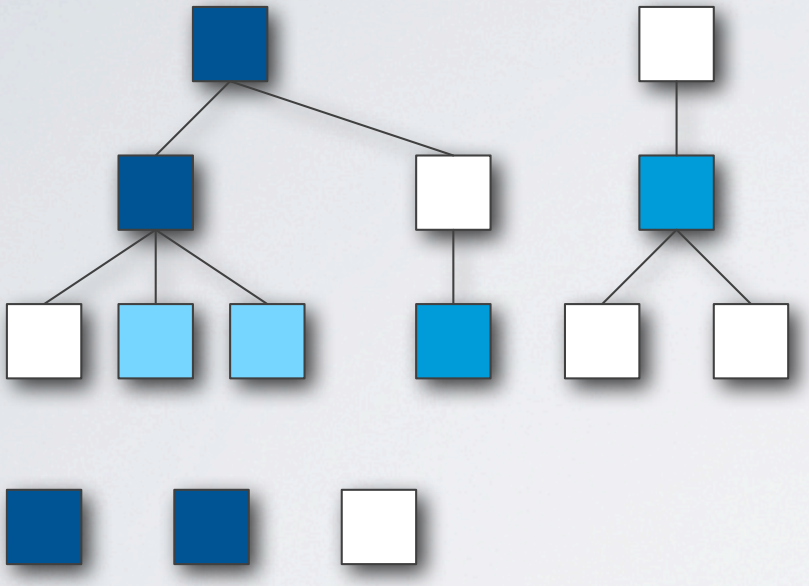
[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Measuring the popularity of classes

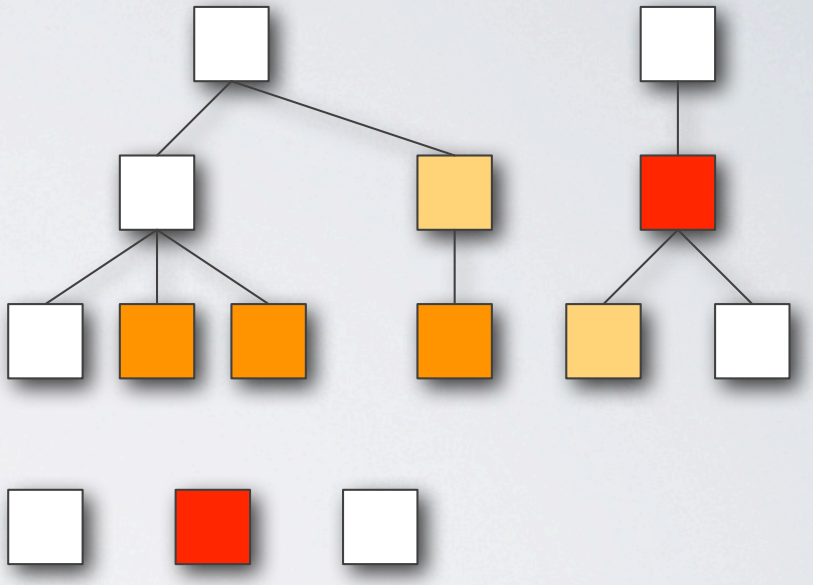


[Moose-dev] 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] Re: 2 more bugs in the MooseFinder	Tudor Girba
[Moose-dev] Re: 2 more bugs in the MooseFinder	Simon Denier
[Moose-dev] Re: 2 more bugs in the MooseFinder	Alexandre Bergel
[Moose-dev] petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Alexandre Bergel
[Moose-dev] Re: petit parser	Lukas Renggli
[Moose-dev] Re: petit parser	Tudor Girba
[Moose-dev] Re: petit parser	Simon Denier
[Moose-dev] Re: petit parser	Stéphane Ducasse
[Moose-dev] Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Simon Denier
[Moose-dev] Re: Missing dependencies?	Tudor Girba
[Moose-dev] Re: Missing dependencies?	Alexandre Bergel
[Moose-dev] Re: Missing dependencies?	Stéphane Ducasse
[Moose-dev] TheMooseBook?	Simon Denier
[Moose-dev] Re: TheMooseBook?	Tudor Girba
[Moose-dev] Fwd: [Pharo-project] Sprint at Bern	Stéphane Ducasse

Using popularity metrics to predict defects

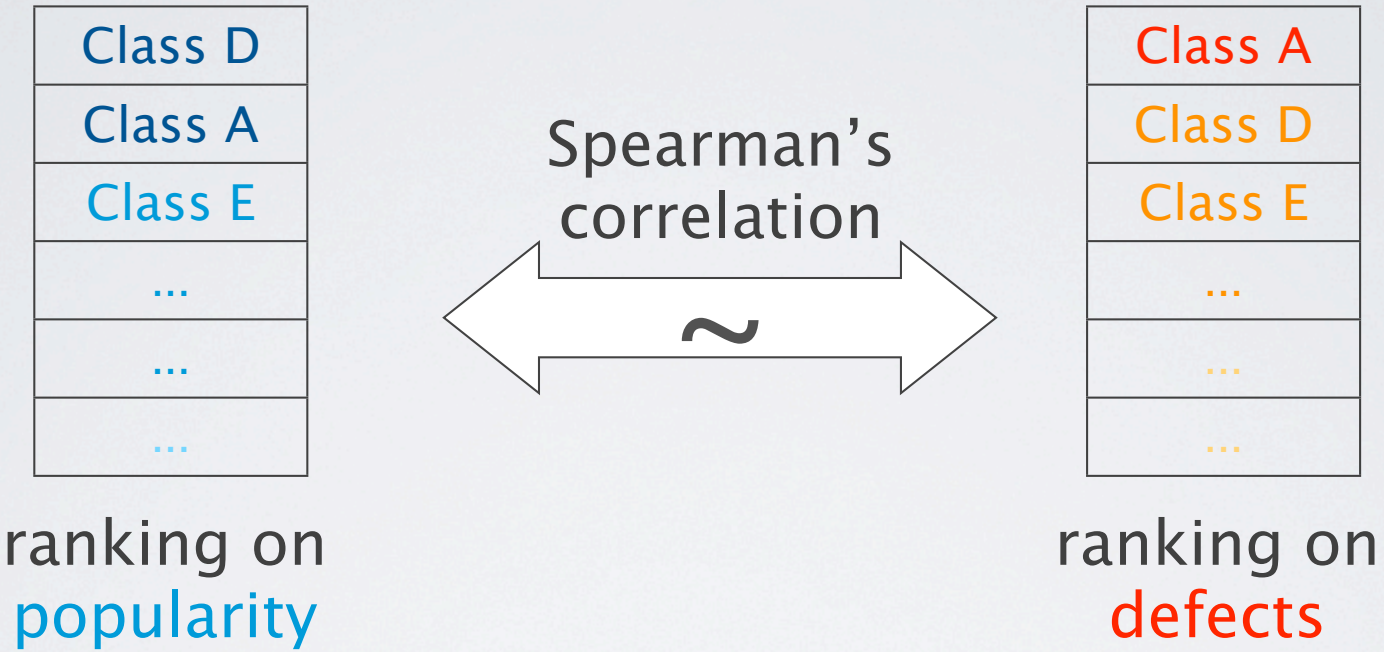


popularity of classes

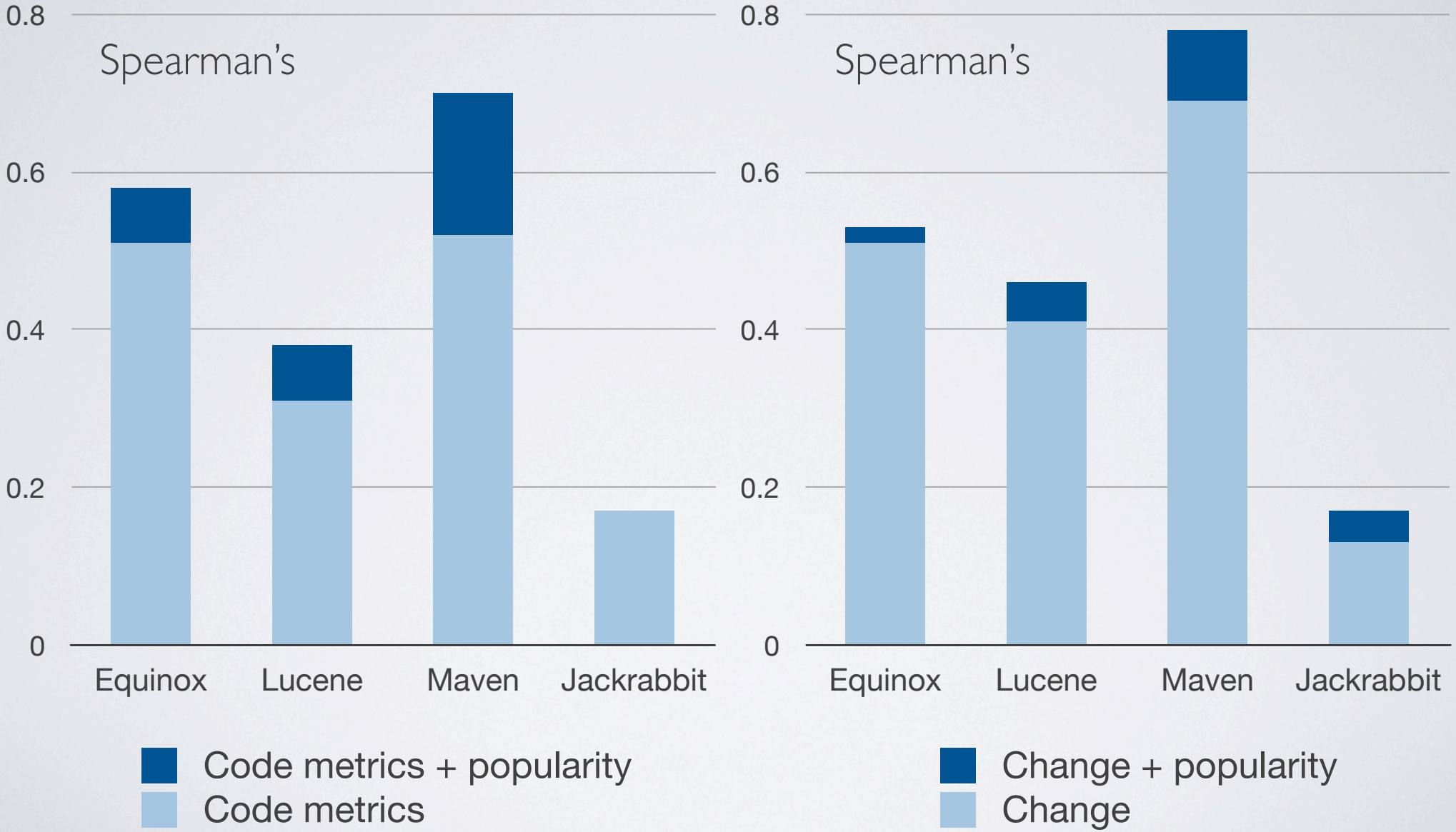


number of defects

Using popularity metrics to predict defects



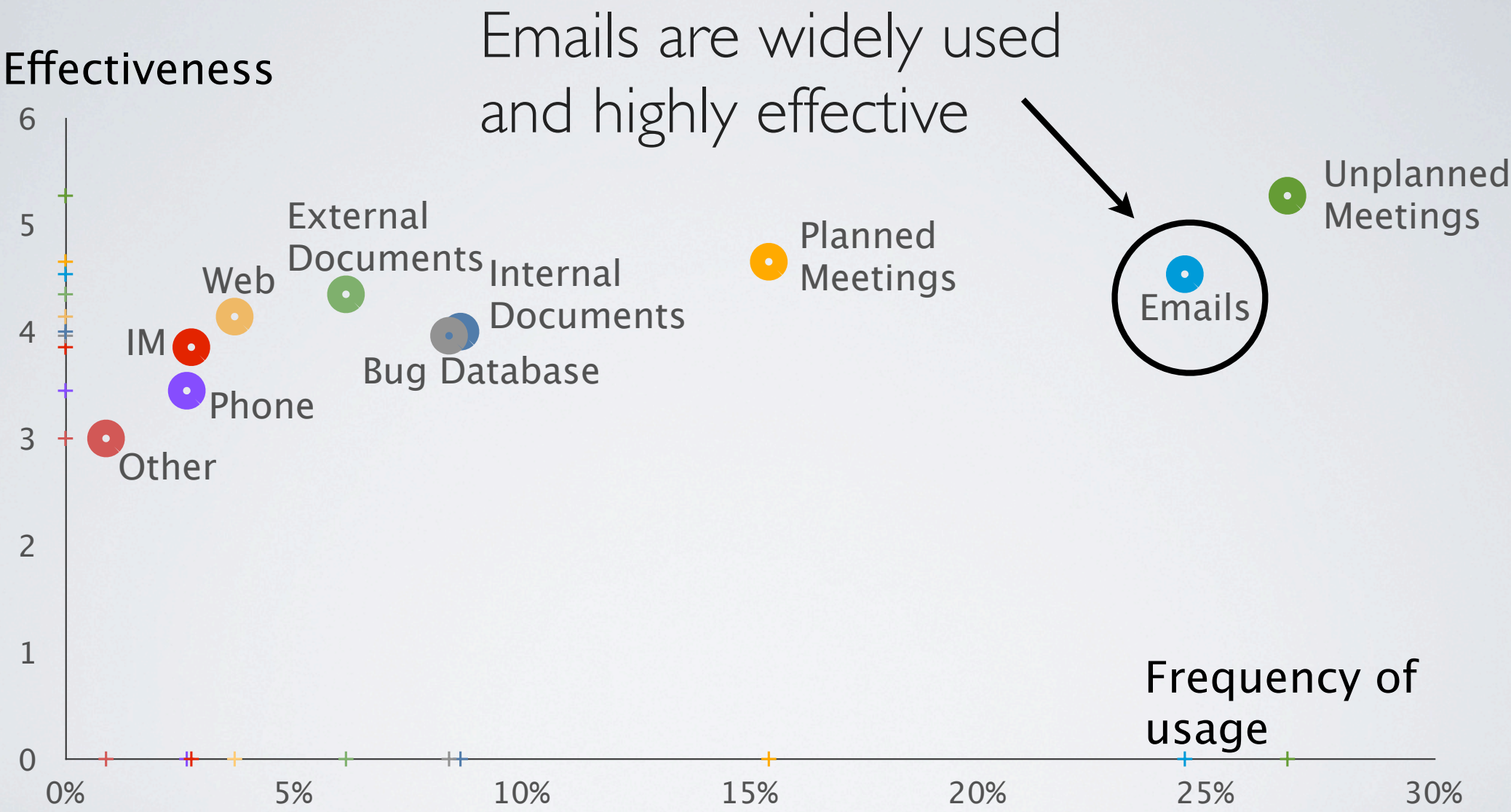
Popularity metrics improve defect prediction



Emails are important for software development

- The analysis of "facts" (e.g. SCM meta-data) points us to code entities that change the most
- The analysis of "documents" (e.g. email) helps us to understand why they are changed

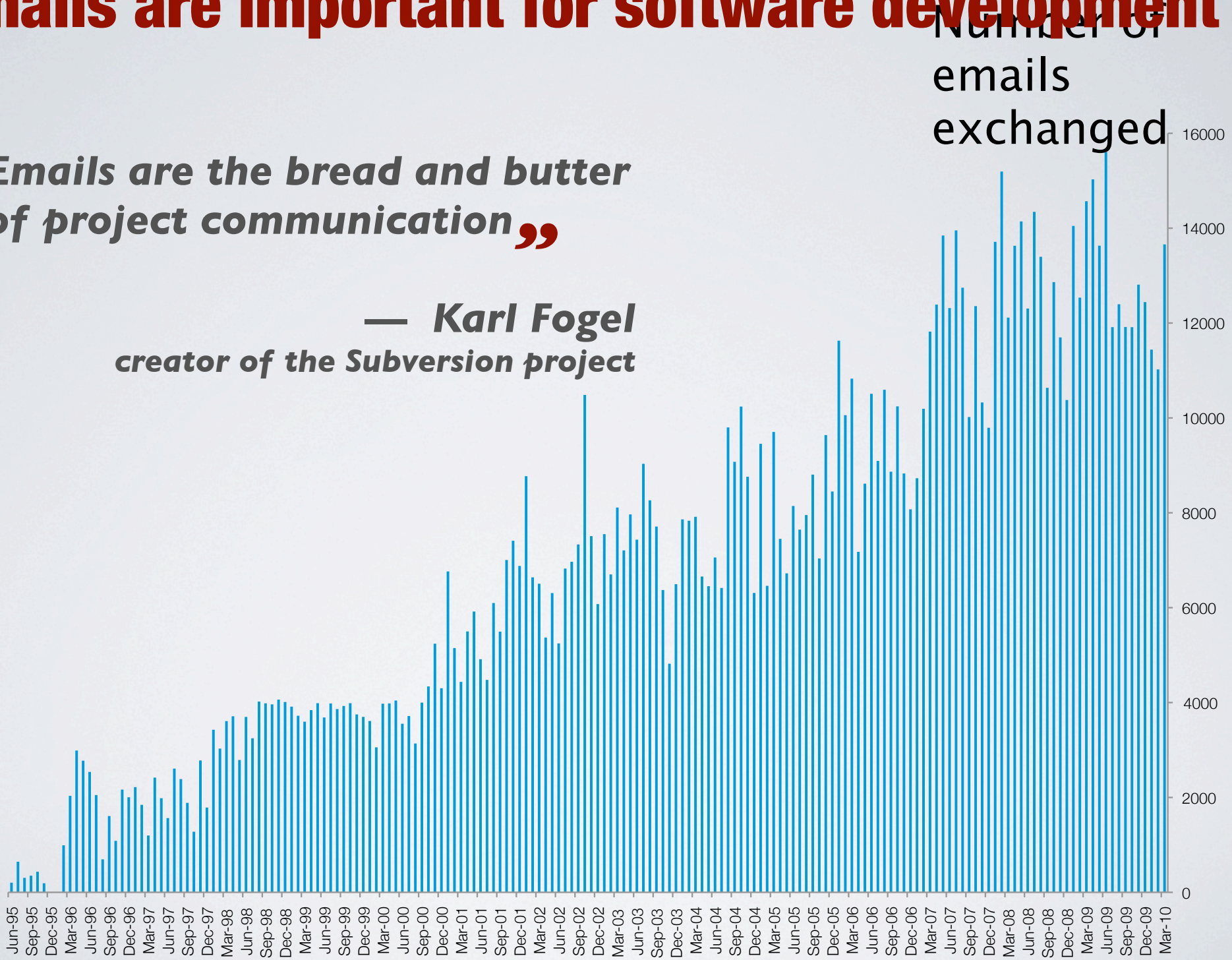
Emails are important for software development



LaToza et al., *Maintaining mental models: a study of developer work habits*, ICSE 2006

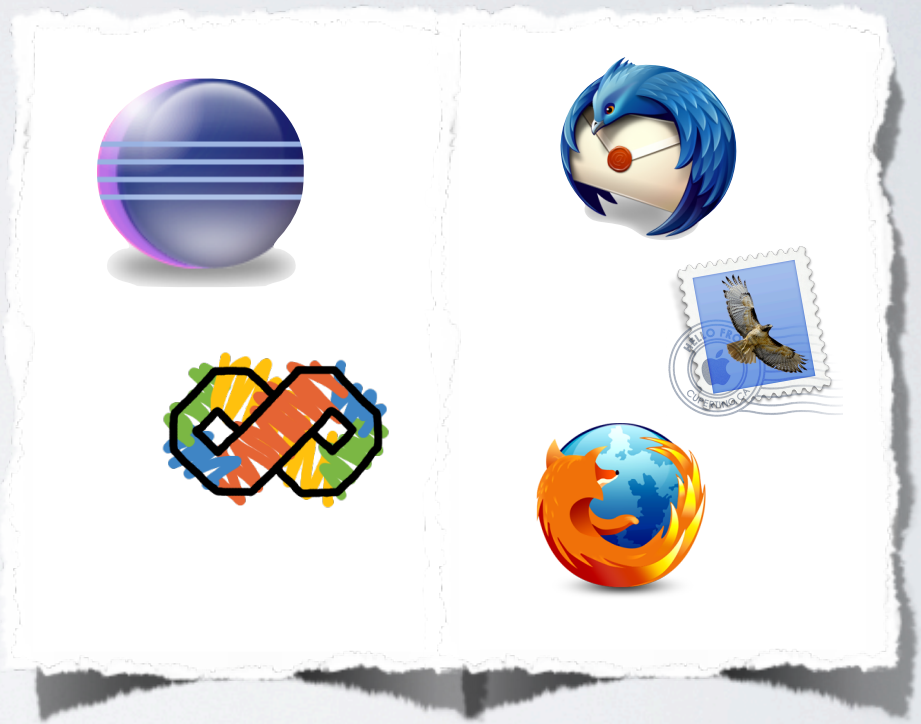
Emails are important for software development

> *"Emails are the bread and butter of project communication"*
>
> — *Karl Fogel*
> *creator of the Subversion project*



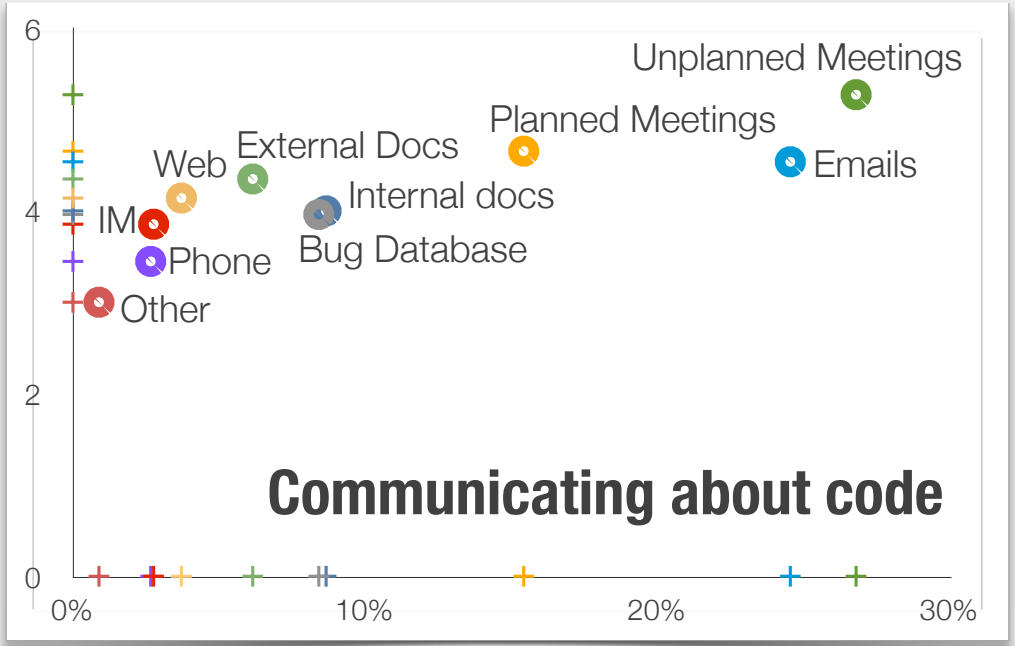
Emails are difficult to use

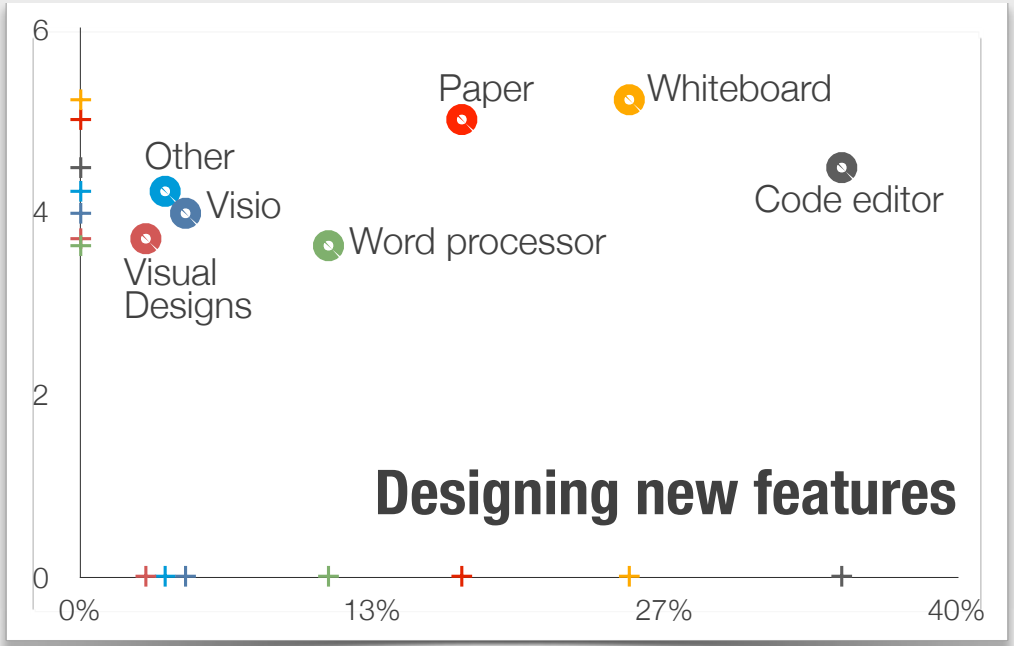
Information overload

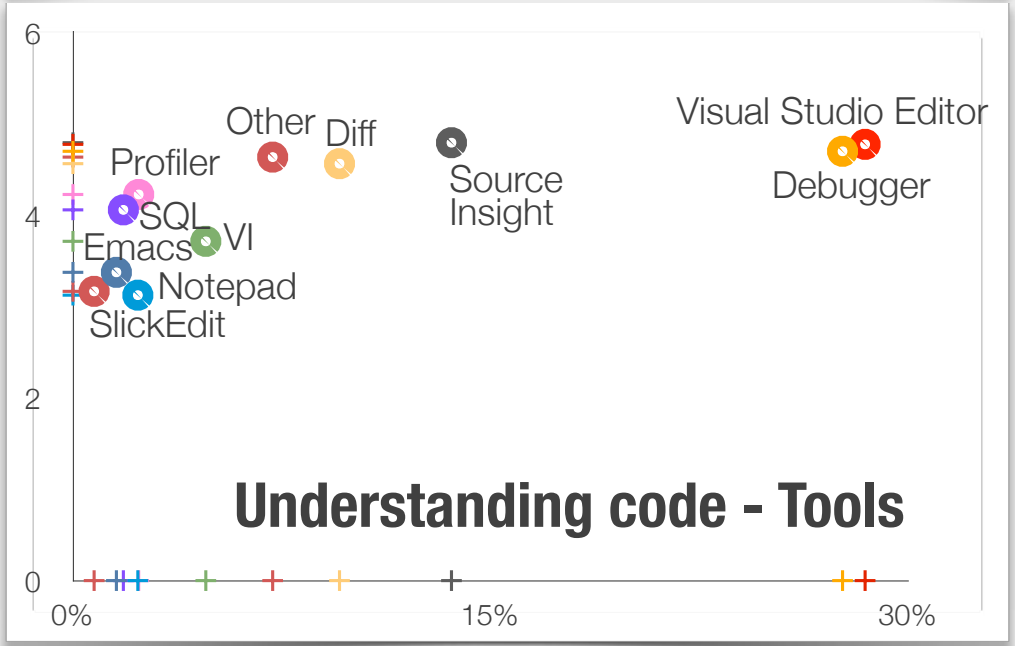


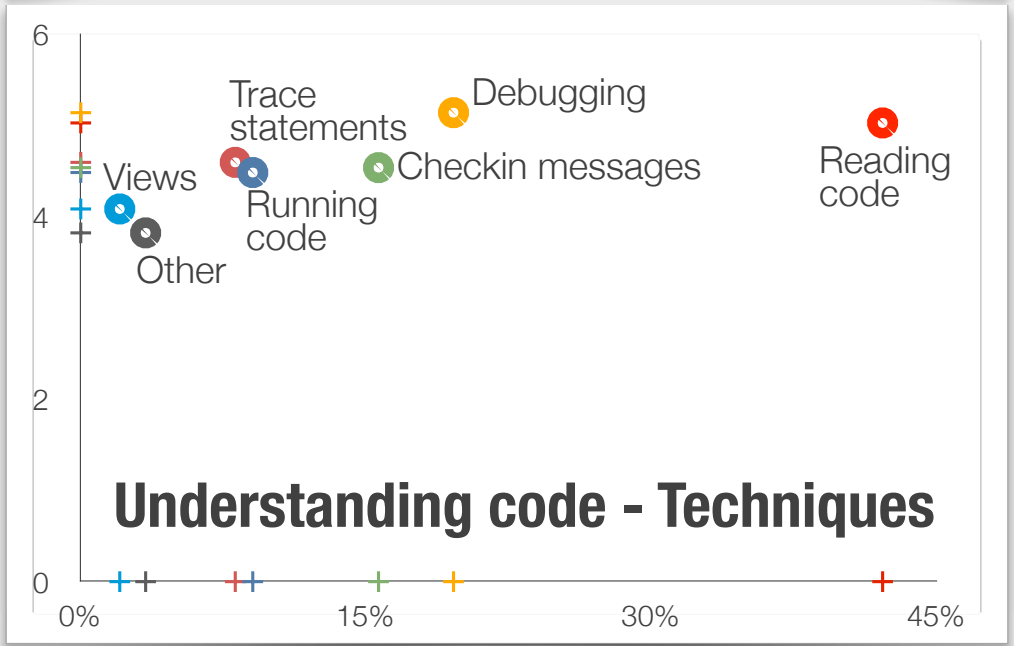
Separation from
actual development

Emails are difficult to use



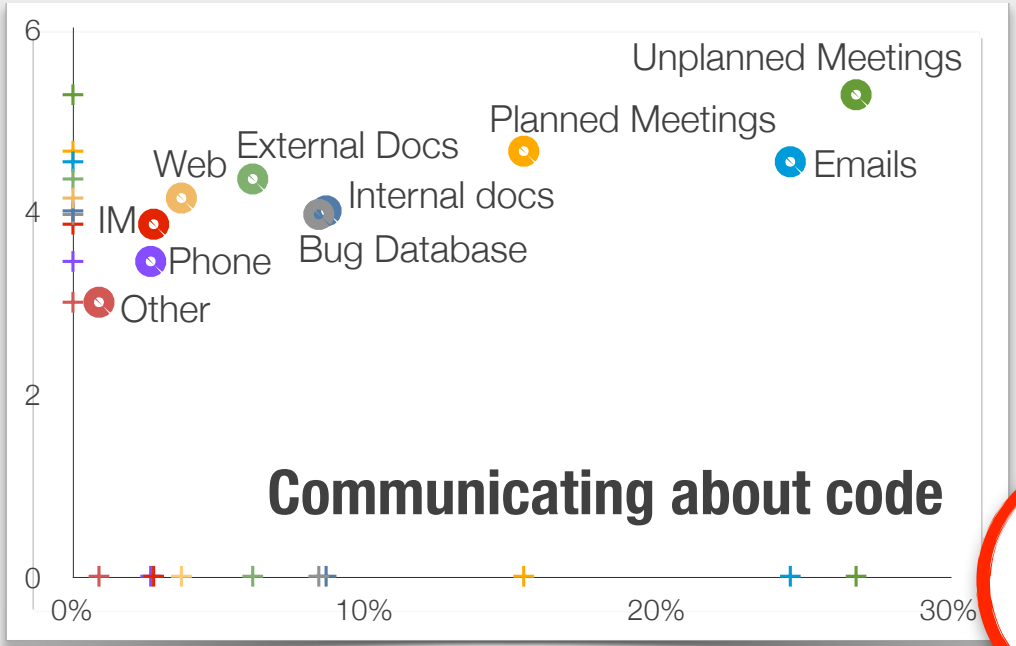


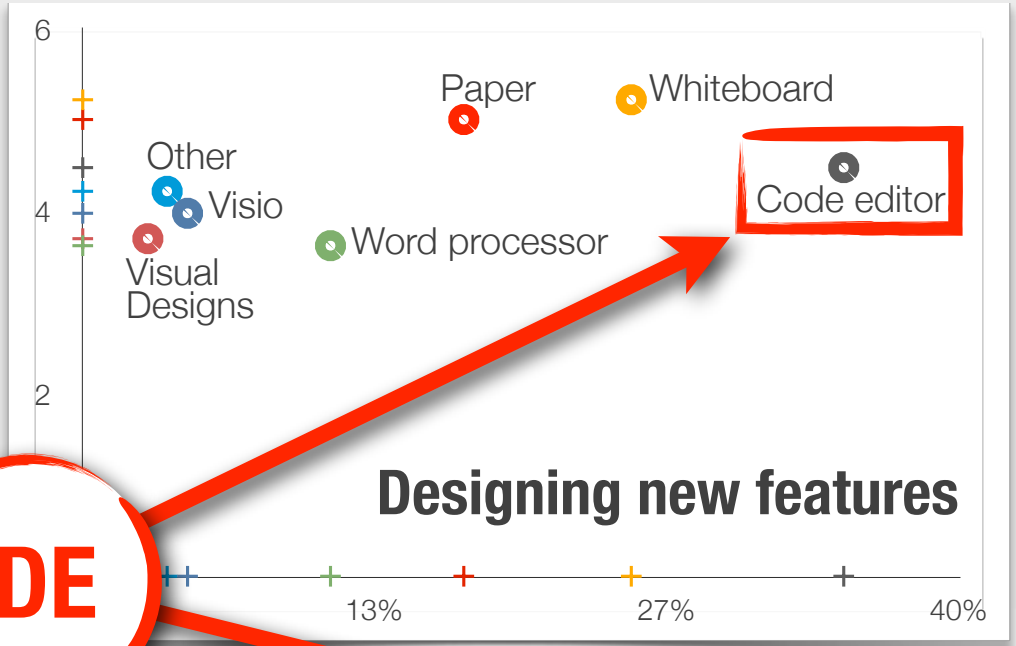




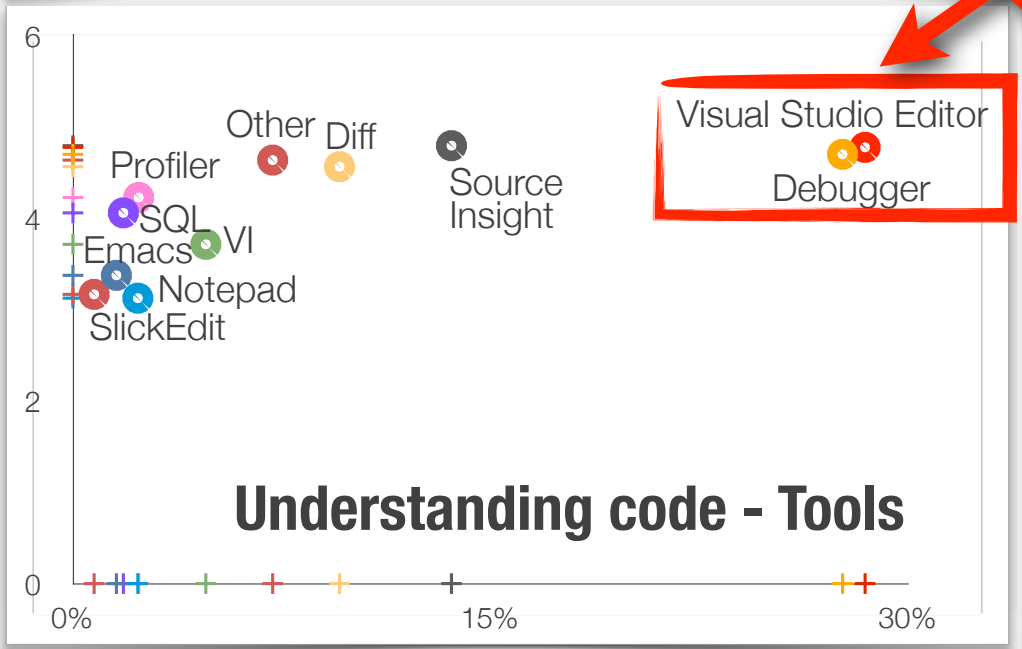
LaToza et al., *Maintaining mental models: a study of developer work habits*, ICSE 2006

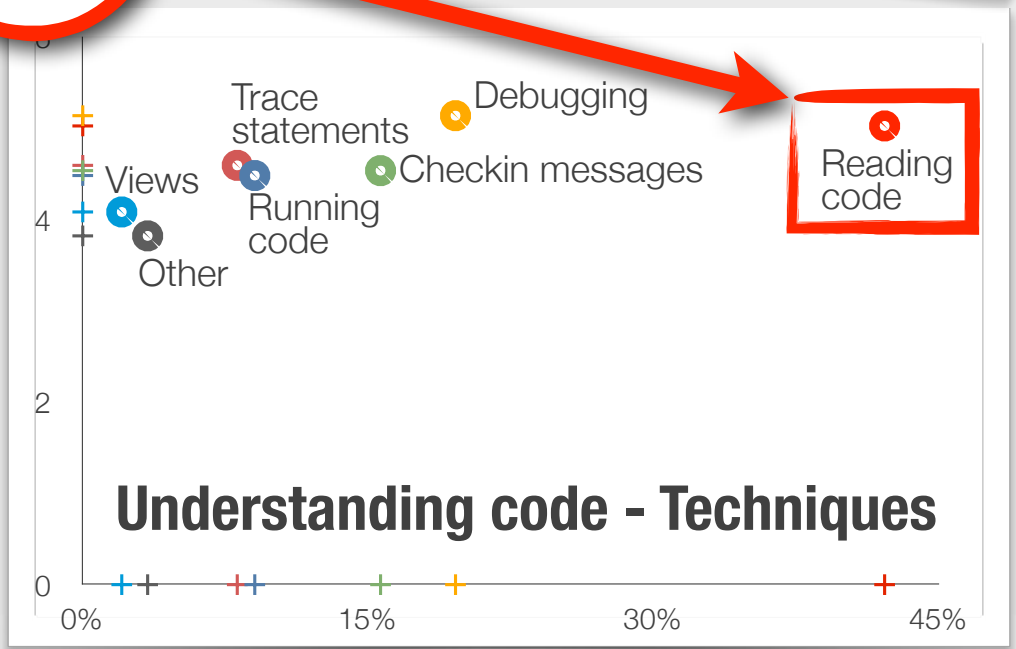
Emails are difficult to use





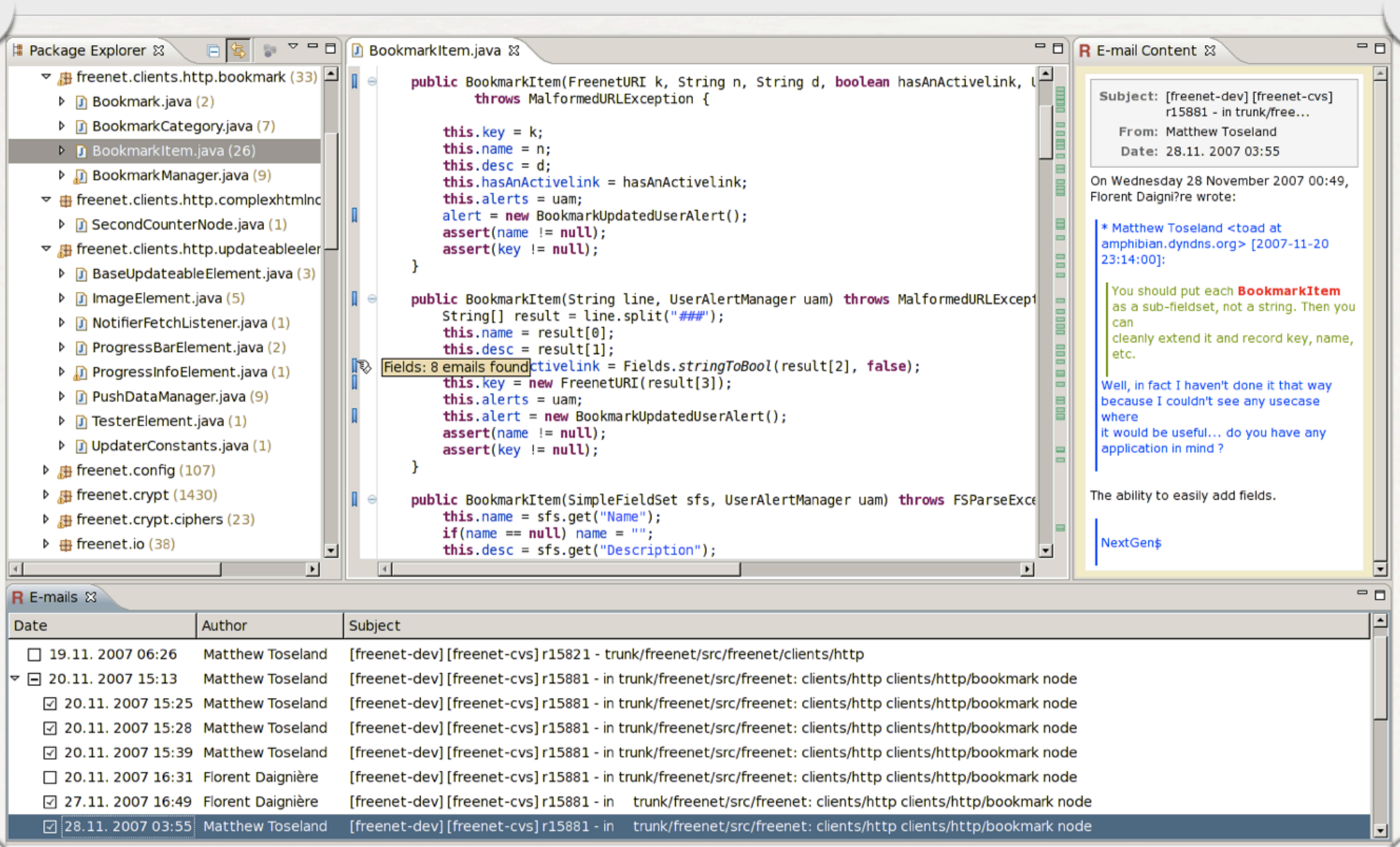
IDE





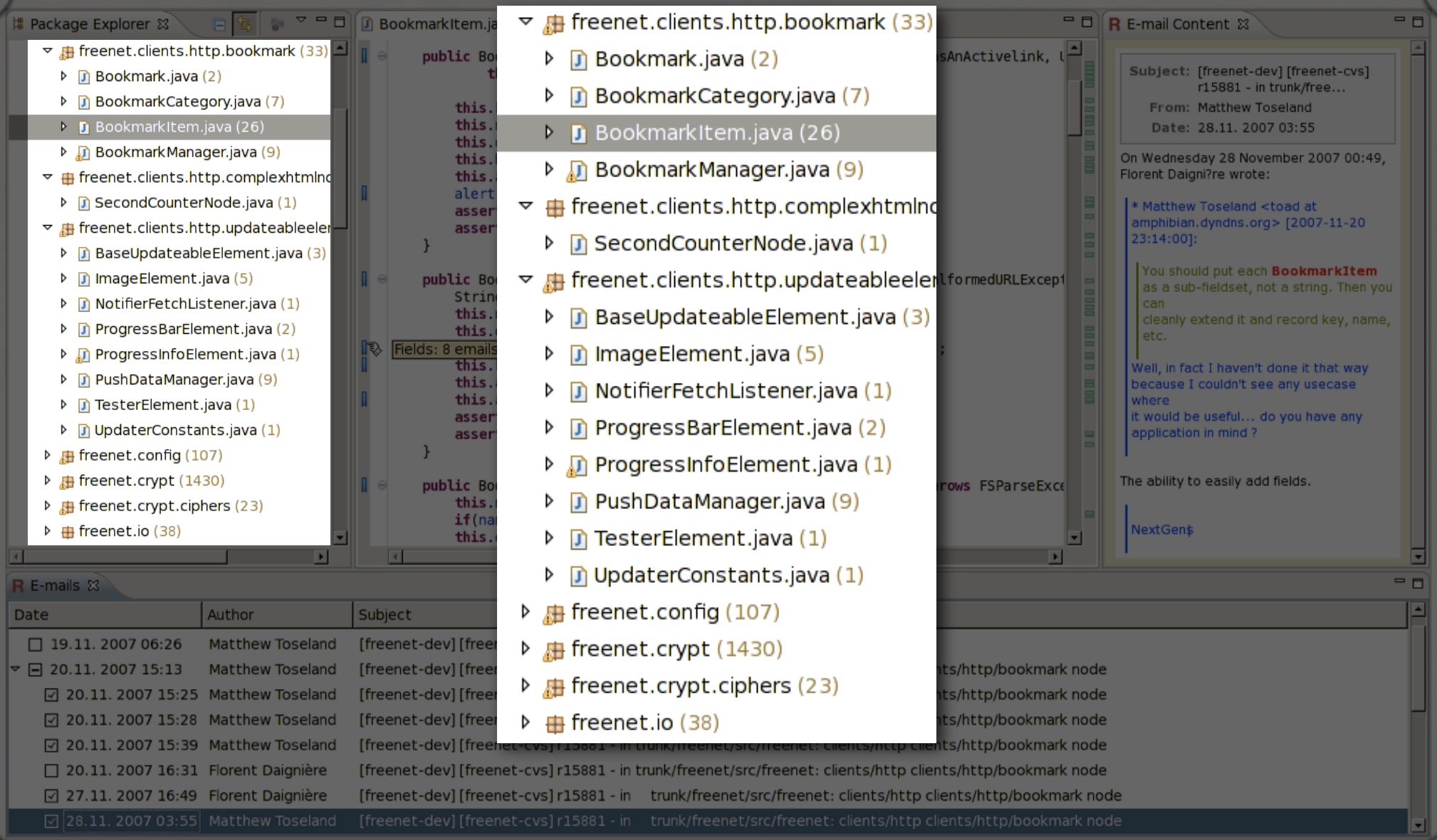
LaToza et al., *Maintaining mental models: a study of developer work habits*, ICSE 2006

Integrating emails into the IDE



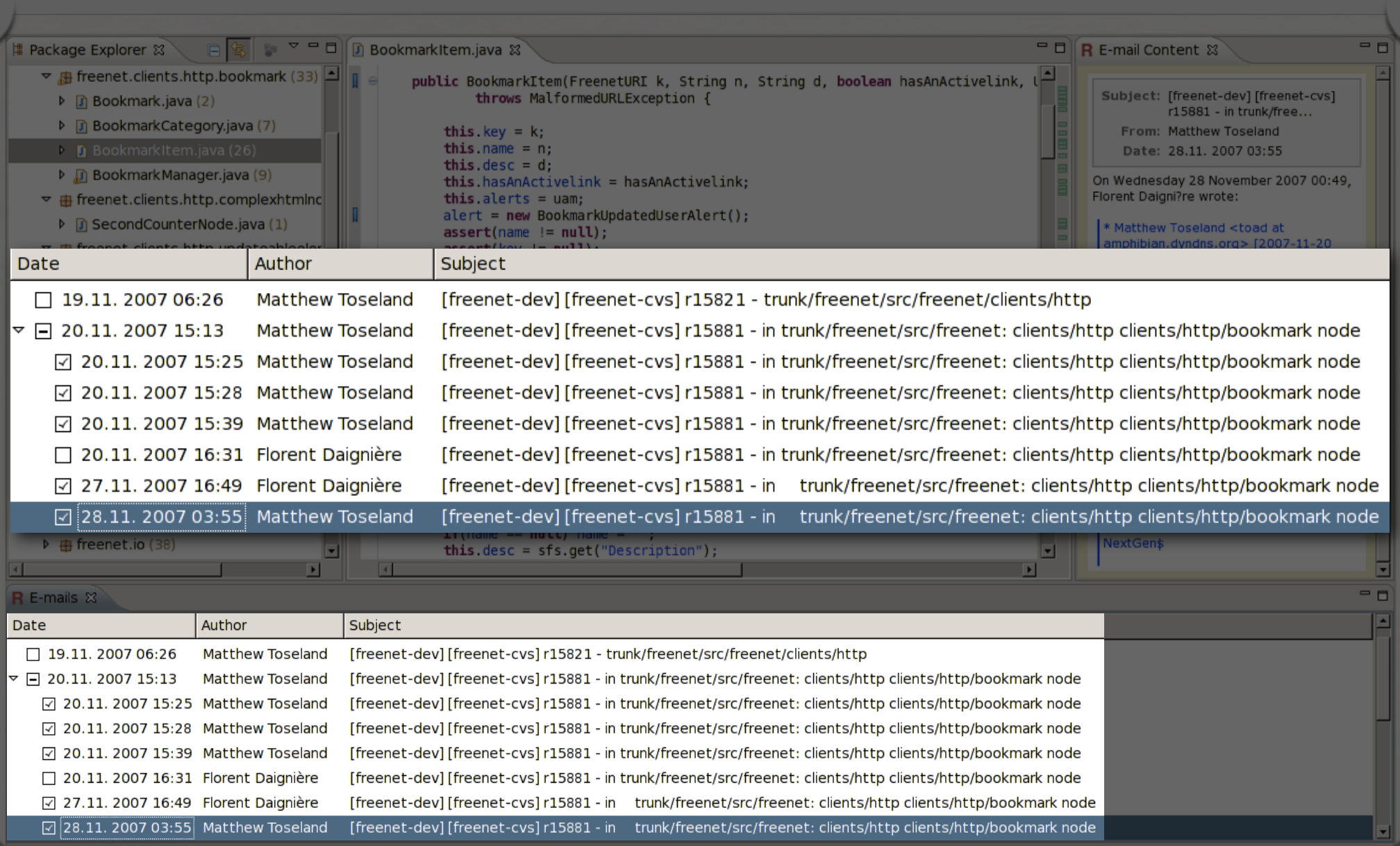
Bacchelli et al., *RTFM (Read the Factual Mails) - Augmenting Program Comprehension with Remail*, CSMR 2011

Integrating emails into the IDE



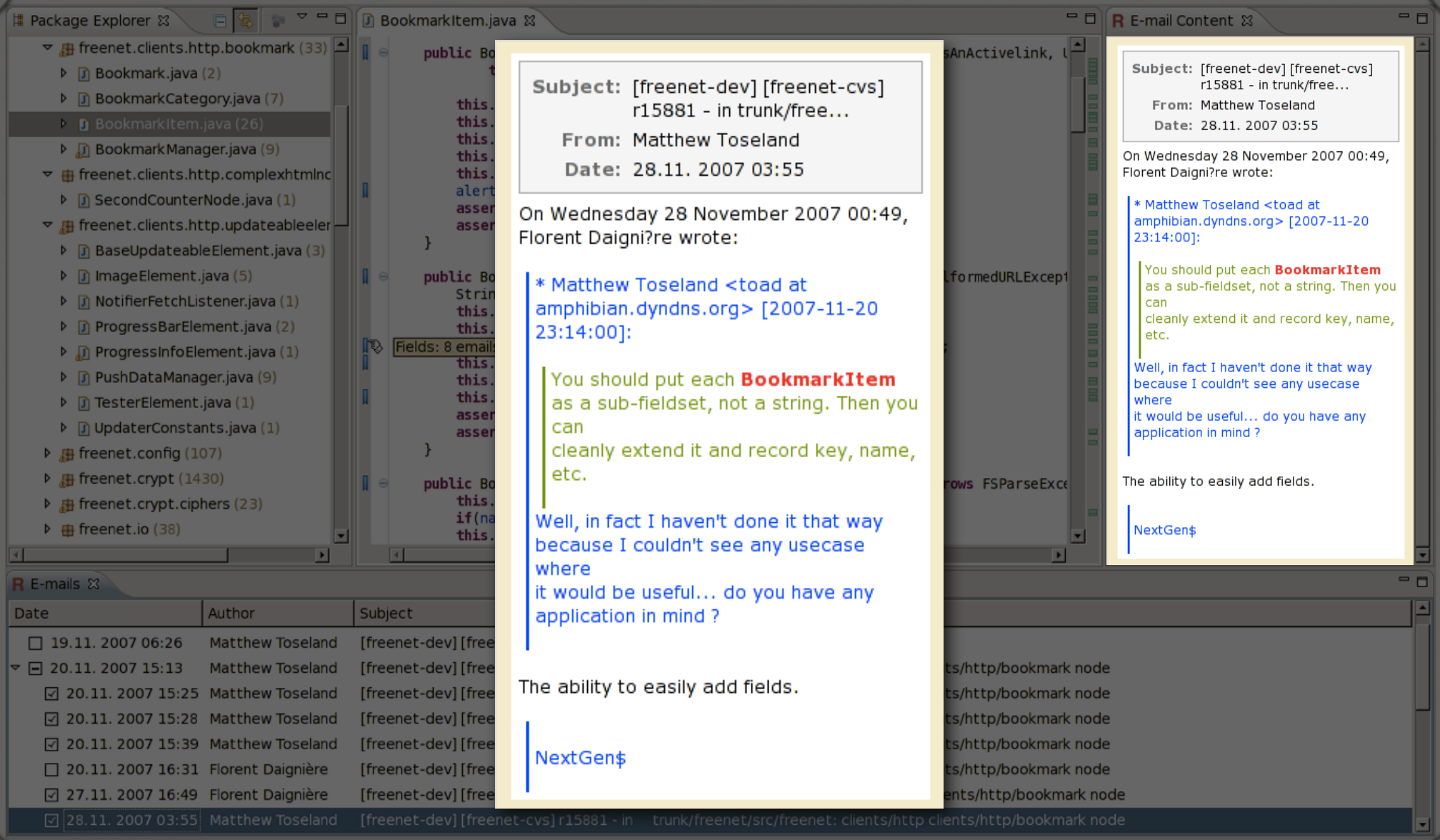
Bacchelli et al., *RTFM (Read the Factual Mails) - Augmenting Program Comprehension with Remail*, CSMR 2011

Integrating emails into the IDE



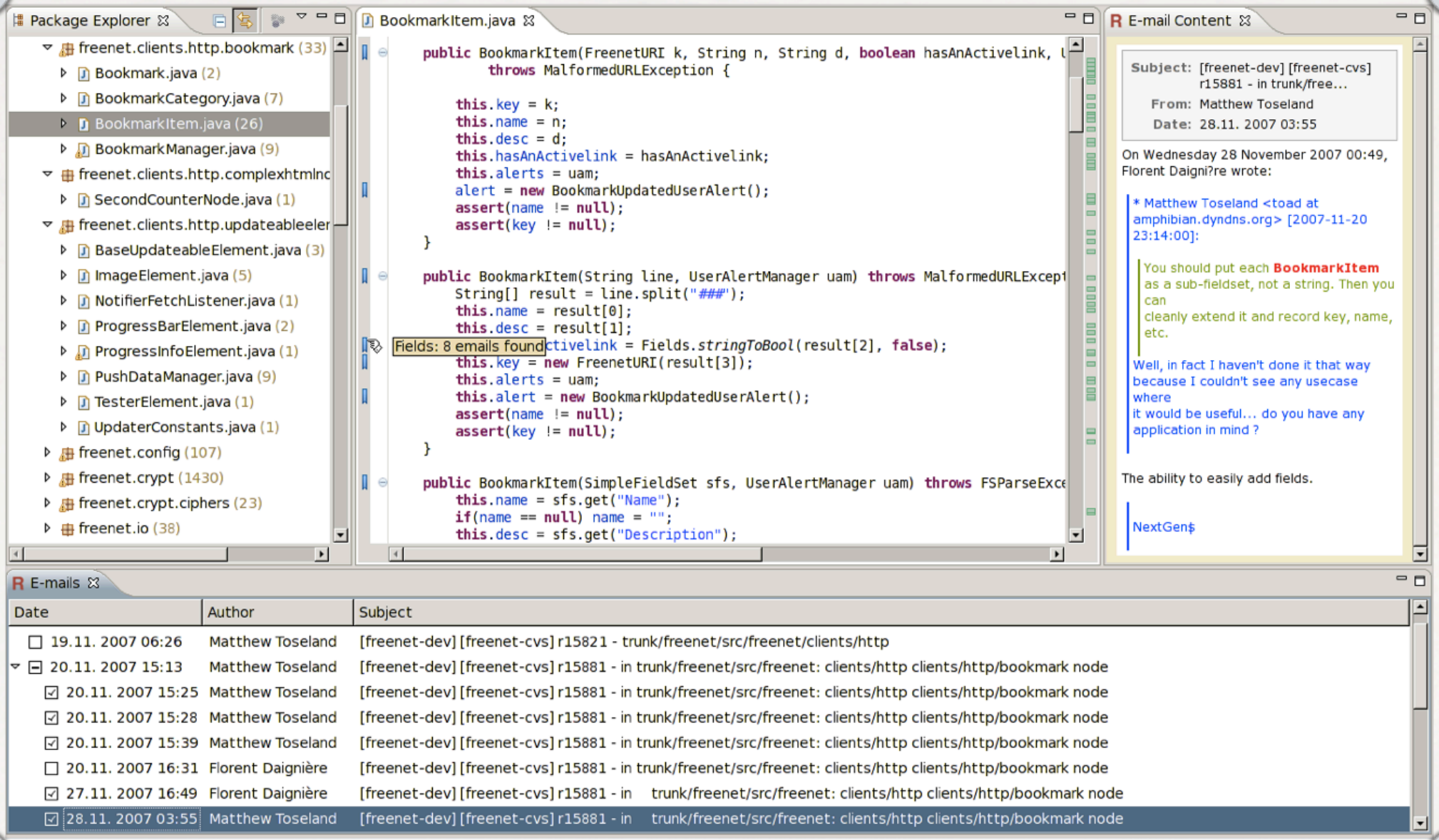
Bacchelli et al., *RTFM (Read the Factual Mails) - Augmenting Program Comprehension with Remail*, CSMR 2011

Integrating emails into the IDE



Bacchelli et al., *RTFM (Read the Factual Mails) - Augmenting Program Comprehension with Remail*, CSMR 2011

Integrating emails into the IDE



Bacchelli et al., *RTFM (Read the Factual Mails) - Augmenting Program Comprehension with Remail*, CSMR 2011

Using REMail for program comprehension

- Finding entry points in a software system
- Conducting evolution analysis
- Improving expert finding techniques
- Recovering additional documentation about systems' entities

Improving expert finding techniques

ExceptionMonitor

Revision	Date	Author
995,776	Oct 2010	elecharny
900,040	Jan 2010	elecharny
783,334	Jan 2009	elecharny
774,593	May 2009	elecharny
678,335	Jul 2008	mwebb
671,827	Jun 2008	jvermillard
576,217	Sep 2007	trustin
565,669	Aug 2007	trustin
555,855	Jul 2007	trustin
497,314	Jan 2007	trustin

Improving expert finding techniques

ExceptionMonitor

Revision	Date	Author
995,776	Oct 2010	elecharny
900,040	Jan 2010	elecharny
783,334	Jan 2009	elecharny
774,593	May 2009	elecharny
678,335	Jul 2008	mwebb
671,827	Jun 2008	jvermillard
576,217	Sep 2007	trustin
565,669	Aug 2007	trustin
555,855	Jul 2007	trustin
497,314	Jan 2007	trustin

Experts:

Improving expert finding techniques

ExceptionMonitor

Date	Author	Subject
16.06. 2006 19:24	John Preston	Addition of SocketConnector method to handle socks proxies.
01.11. 2006 01:39	Hieu Phan Thanh	RE: [MINA 0.8.3] Could not stop listening on Port
06.11. 2006 17:07	Trustin Lee	Re: [MINA 1.0.0] Could not stop listening on Port
28.07. 2007 03:07	James Im	Rationale for Thread.sleep(1000) in Worker ?
23.08. 2007 06:07	Pierre-Louis Bonicoli	Re: client doesn't stop
05.02. 2008 07:45	Sangjin Lee	Re: connect timeout
10.06. 2008 04:31	Julien Vermillard	Re: [2.0 refactoring] Reviewing core packages was : [2.0 refactoring]
20.06. 2008 05:20	Emmanuel Lecharny	Re: [2.0 refactoring] Reviewing core packages was : [2.0 refactoring]
03.07. 2008 09:45	Emmanuel Lecharny	Is the ExceptionMonitor usefull ?
05.07. 2008 06:27	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
05.07. 2008 10:50	peter royal	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 06:01	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 06:07	Emmanuel Lecharny	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 22:30	Emmanuel Lecharny	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 12:42	Adam Fisk	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 13:39	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 13:45	Mark Webb	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 14:30	Niklas Gustavsson	Re: Is the ExceptionMonitor usefull ?
11.08. 2008 23:35	Julien Vermillard	Release of Apache MINA 2.0.0-M3
09.06. 2009 01:37	Irving, Dave	RE: Exceptions in MINA
09.06. 2009 11:36	Michael Ossareh	Re: Exceptions in MINA

Experts: elecharny mwebb jvermillard trustin

Improving expert finding techniques

ExceptionMonitor

Date	Author	Subject
16.06. 2006 19:24	John Preston	Addition of SocketConnector method to handle socks proxies.
01.11. 2006 01:39	Hieu Phan Thanh	RE: [MINA 0.8.3] Could not stop listening on Port
06.11. 2006 17:07	Trustin Lee	Re: [MINA 1.0.0] Could not stop listening on Port
28.07. 2007 03:07	James Im	Rationale for Thread.sleep(1000) in Worker ?
23.08. 2007 06:07	Pierre-Louis Bonicoli	Re: client doesn't stop
05.02. 2008 07:45	Sangjin Lee	Re: connect timeout
10.06. 2008 04:31	Julien Vermillard	Re: [2.0 refactoring] Reviewing core packages was : [2.0 refactoring]
20.06. 2008 05:20	Emmanuel Lecharny	Re: [2.0 refactoring] Reviewing core packages was : [2.0 refactoring]
03.07. 2008 09:45	Emmanuel Lecharny	Is the ExceptionMonitor useful ?
05.07. 2008 06:27	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
05.07. 2008 10:50	peter royal	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 06:01	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 06:07	Emmanuel Lecharny	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 22:30	Emmanuel Lecharny	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 12:42	Adam Fisk	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 13:39	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 13:45	Mark Webb	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 14:30	Niklas Gustavsson	Re: Is the ExceptionMonitor usefull ?
11.08. 2008 23:35	Julien Vermillard	Release of Apache MINA 2.0.0-M3
09.06. 2009 01:37	Irving, Dave	RE: Exceptions in MINA
09.06. 2009 11:36	Michael Ossareh	Re: Exceptions in MINA

karasul

irving

Experts: elecharny mwebb jvermillard trustin

Recovering additional documentation

CircularQueue

"unbounded circular queue based on array."

—**CircularQueue,** code comment

Recovering additional documentation

CircularQueue

> "Even though ConcurrentLinkedQueue performs bad comparing to synchronized CircularQueue, the former is not only a comparable data structure, but it's also thread safe, and tested. [We should] remove the CircularQueue from the code base."
>
> **— About CircularQueue,** email thread

Suggesting important email discussions

- Development emails are important, but since they come in sheer volume, it is easy to miss important discussions
- Ibrahim, et al. tackled this problem by
  - investigating which are the main factors that encourage developers to contribute to mailing lists
  - building models that identify discussion threads to which a developer should contribute

Ibrahim, et al., *Should I contribute to this discussion?* MSR 2010

Analyzed factors categorized in 4 dimensions

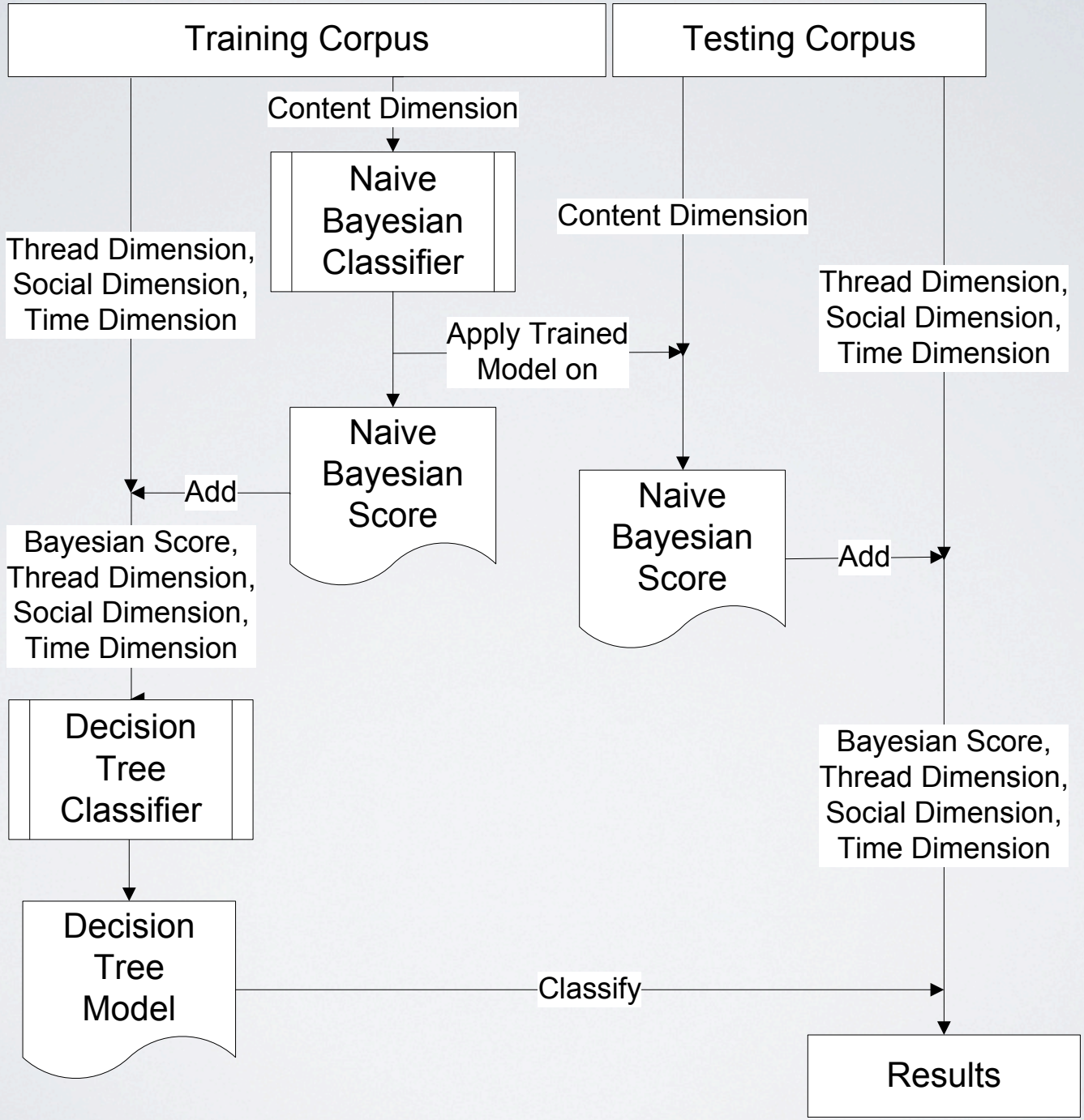
- Thread: length (#emails) and size (#words) of the thread
- Social: do I know the starter of the thread? And the last who posted? How many thread did I contribute to?
- Time: when the emails were posted to the thread
- Content: content of the thread

Analyzed factors categorized in 4 dimensions

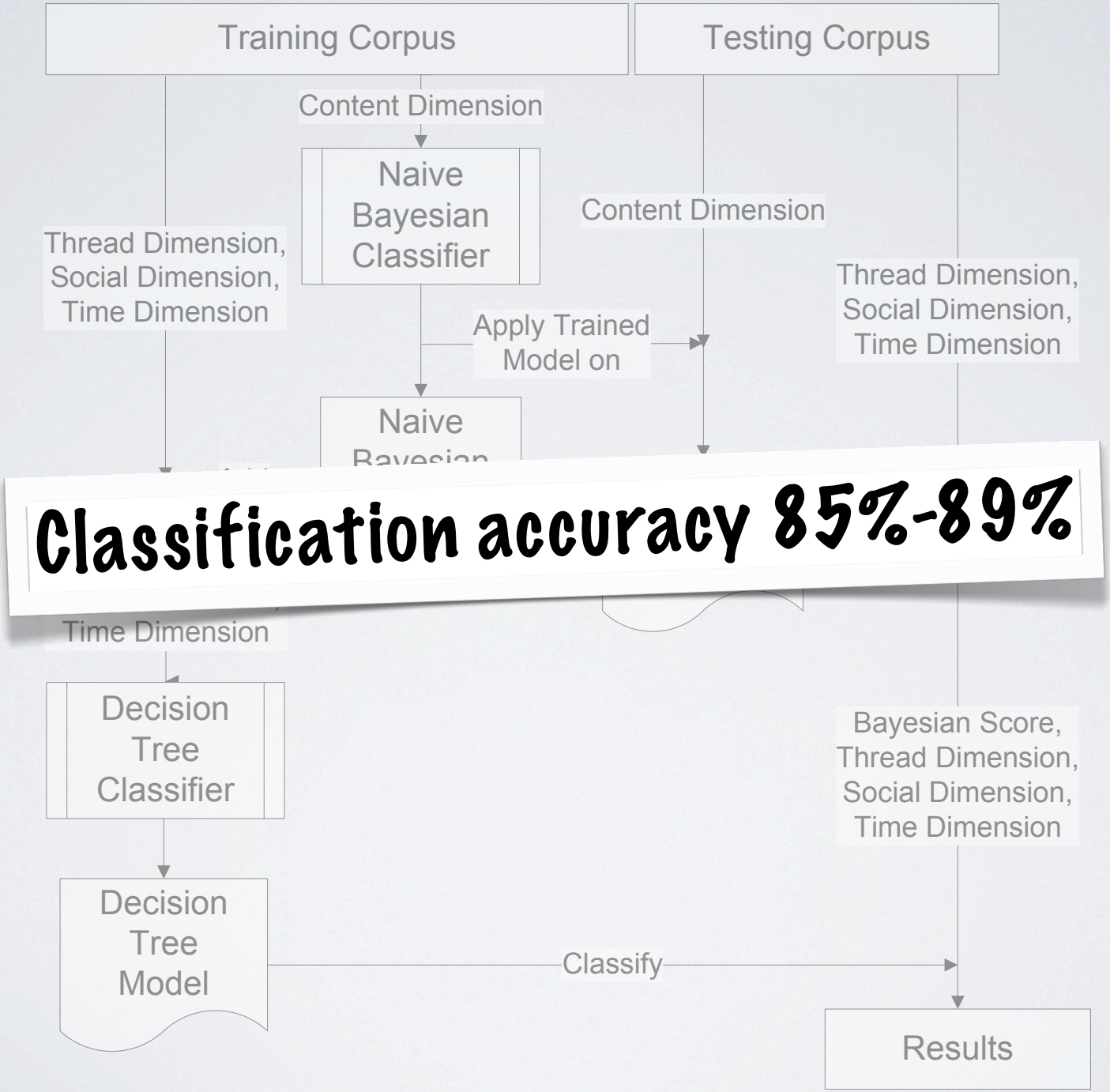
- Thread: length (#emails) and size (#words) of the thread
- Social: do I know the starter of the thread? And the last who posted? How many thread did I contribute to?
- Time: when the emails were posted to the thread
- Content: content of the thread

Most important factors

Identifying relevant discussion threads



Identifying relevant discussion threads



Human factors

- Goal: Extract, measure and analyze communication and organizational information to
  - improve previous MSR techniques
  - support software maintenance and development
  - perform empirical studies


Defect prediction
Change analysis and prediction
Empirical studies
Expertise & bug triaging
Visual evolution analysis
Human factors

MSR Approaches