2IS55 Software Evolution

What can we learn from version control systems?

Alexander Serebrenik

Technische Universiteit Eindhoven University of Technology

Where innovation starts

Assignment

Assignment 1:

- **Grades: 3 3 1 4 2 2**
  - **Too big, too boring, report writing is a waste of time**

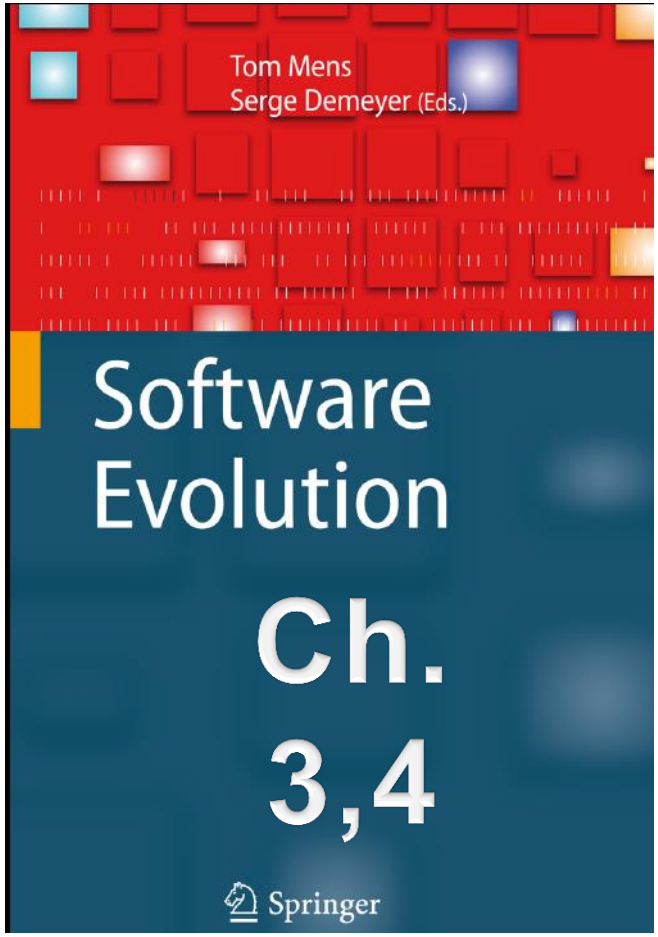
Assignment 2:

- **Grades: 3 1 2 2**
  - **reverse engineering is fun, Rascal as a really working DSL, incremental approach**
  - **Rascal documentation, OFG documentation, translation from the OFG theory to code**

Assignment 3:

- **Deadline: Monday, March 17**
- **Main challenges**
  - **replication precision**
  - **scalability: at least 50 versions**

Sources


Tom Mens
Serge Demeyer (Eds.)
Software
Evolution
Ch.
3,4
Springer


Effective Mining
of
Software Repositories
Marco D'Ambros
REVEAL group
University of Lugano
Switzerland
Romain Robbes
PLEIAD @ DCC
University of Chile
Chile

Recap: Version control systems

- Centralized vs. distributed
- File versioning (CVS) vs. product versioning

- Record at least
  - File name, file/product version, time stamp, committer
  - Commit message

- What can we learn from this?
  - Humans
  - Files
  - Bugs

TODAY !

What can we learn about files?

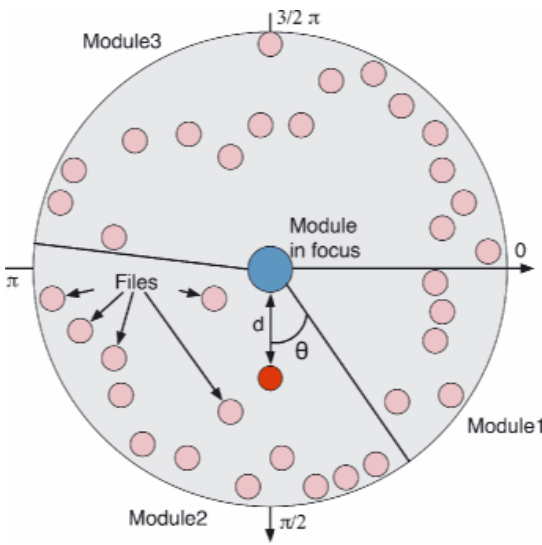
- **Change coupling - two artifacts change together [Ball et al. 1997]**
  - **Based on common commits**
  - **Subversion – easy, CVS – time window**
    - **What about longer transactions?**
    - **Looks like EROSE** (do you still remember?)

What can we learn about files?

- **Change coupling - two artifacts change together [Ball et al. 1997]**
  - **Based on common commits**
  - **Subversion – easy, CVS – time window**
    - **What about longer transactions?**
    - **Looks like EROSE** (do you still remember?)
- **Why change coupling? [D'Ambros, Lanza, Robbes 2009]**
  - **Number of coupled classes (having at least n common commits) correlates with the number of bugs**
    - **Eclipse, $3 \leq n \leq 20$, Spearman $\rho \approx 0.8$**
    - **Mylyn and ArgoUML: Spearman $\rho > 0.5$**
  - **Correlates more with the number of bugs than popular metrics, but less than the number of changes (churn)**

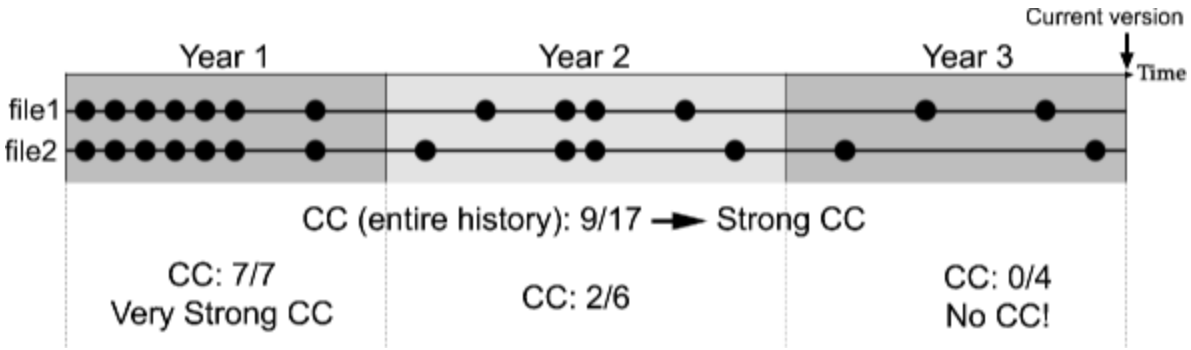
Weapon of choice: Evolution Radar

- **Focuses on one module (component)**
- **Dependencies between the module and other modules (groups of files)**
- **radius d: inverse of change coupling with the closest file of the module in focus**
- **angle θ: certain ordering (alphabetical)**
- **color and size – arbitrary metrics**



Evolution Radar

- **Moving through Time**
  - **Taking entire history into account can be misleading**



  - **Radar is time-dependent: entire history vs. time window**

- **Tracking**
  - **Keep track of a file when Moving through Time**

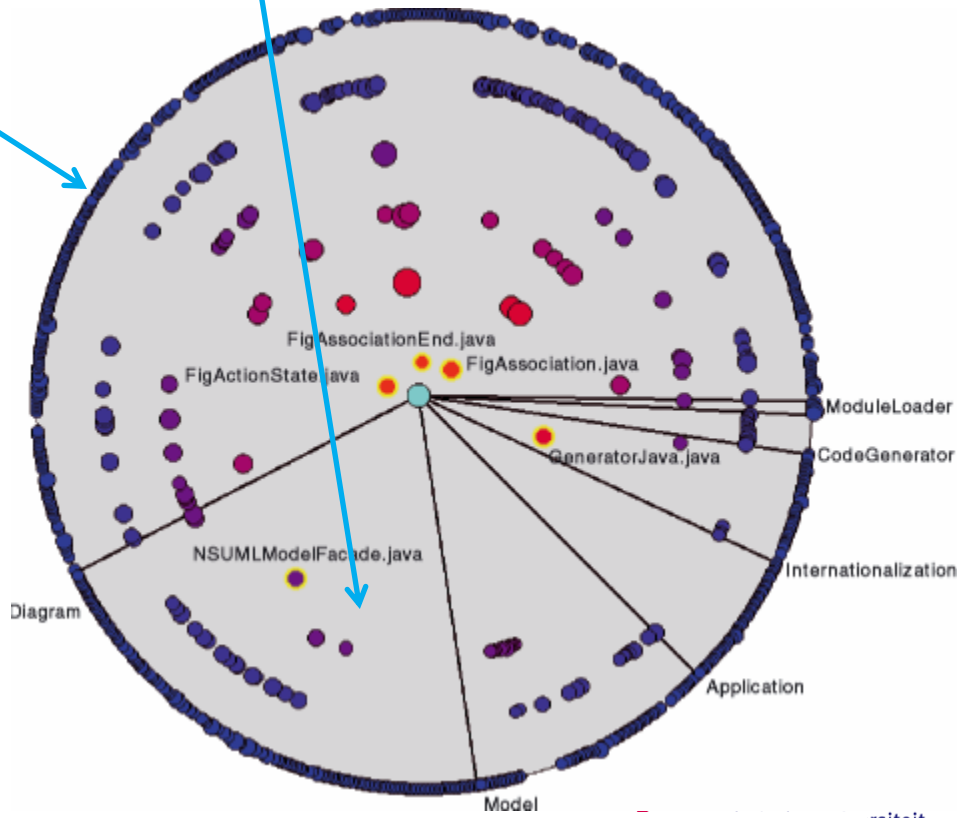
Experiment: ArgoUML

- **Three main components. According to the documentation**
  - **Explorer and Diagram depend on Model**
  - **Explorer and Diagram do not depend on each other**

Experiment: ArgoUML

- Three main components. According to the documentation
  - Explorer and Diagram depend on Model
  - Explorer and Diagram do not depend on each other

- Color: change coupling
- Size: Total number of lines modified during the period
- Focus on Explorer

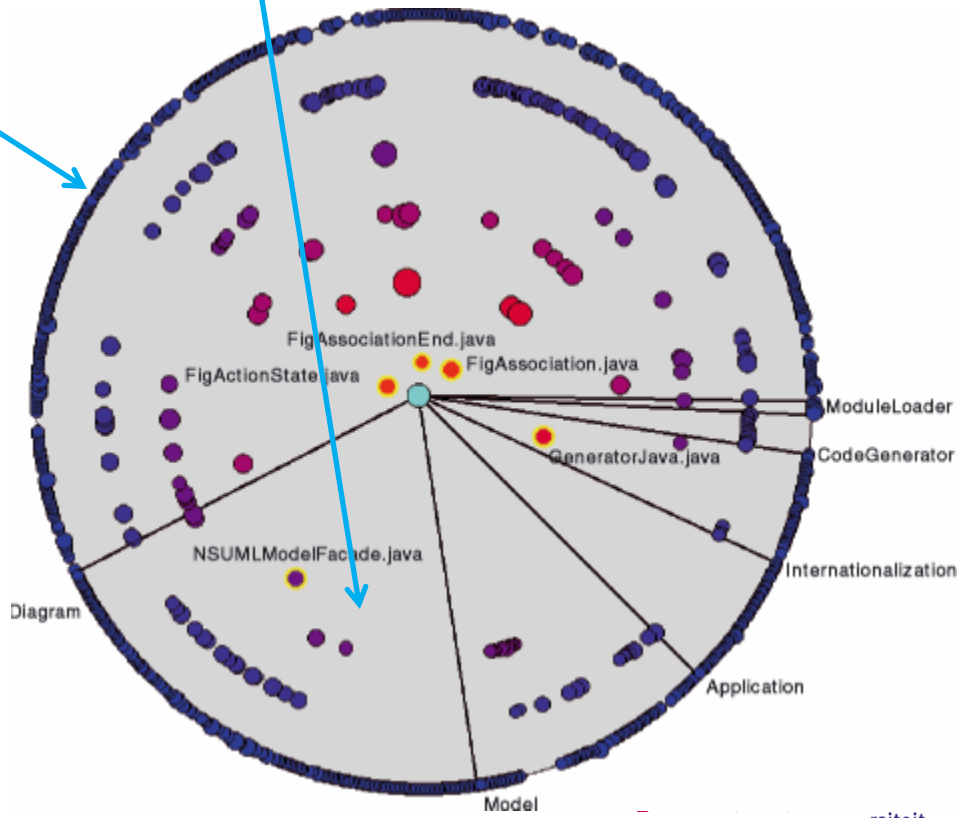


June – December 2005

Experiment: ArgoUML

- Three main components. According to the documentation
  - Explorer and Diagram depend on Model
  - Explorer and Diagram do not depend on each other

- Color: change coupling
- Size: Total number of lines modified during the period
- Focus on Explorer

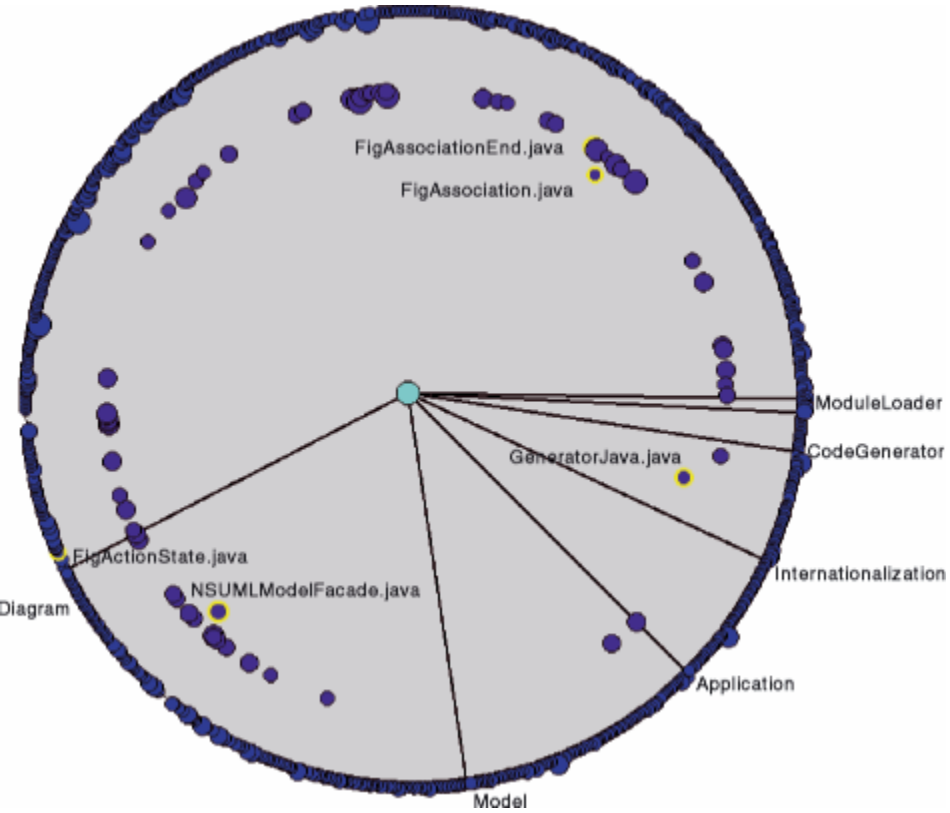


Conclusion: Explorer strongly depends on Diagram

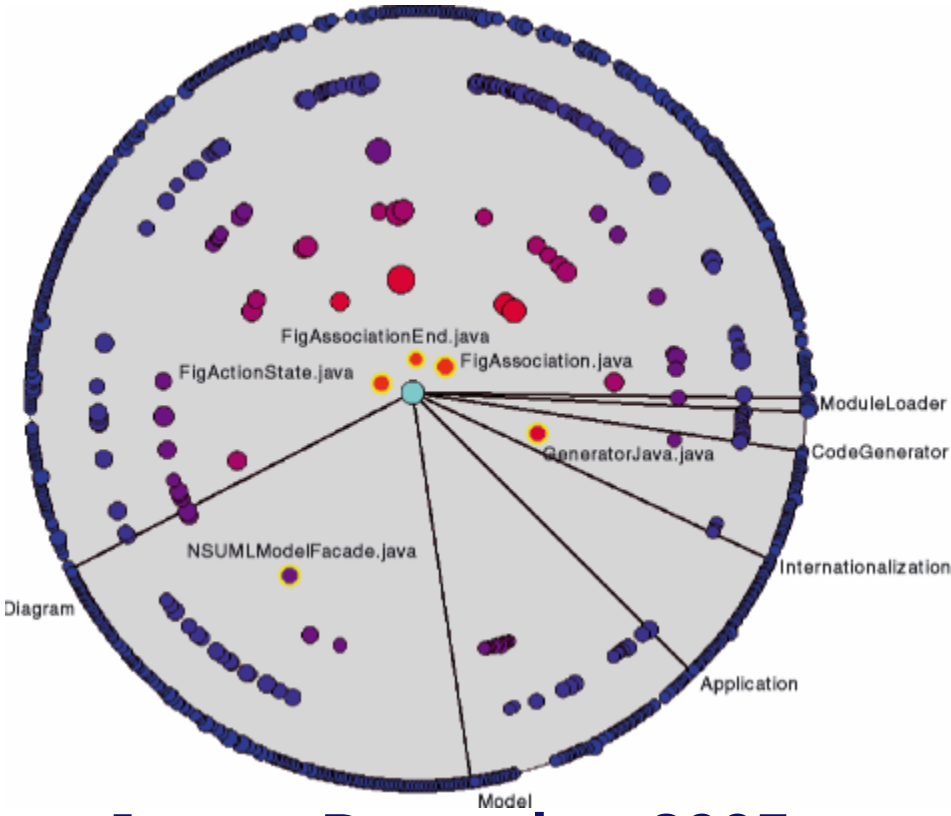
June – December 2005

ArgoUML

- **Fig*.java moved closer to the center: CC increased!**
- **Generator.java is an outlier**



January – June 2005

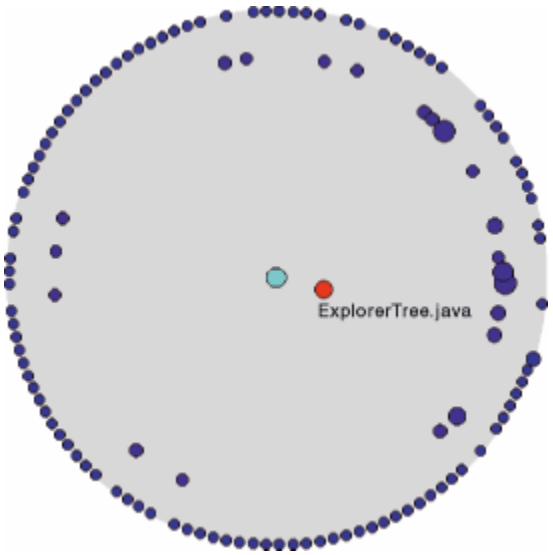


June – December 2005

Technische Universiteit Eindhoven University of Technology

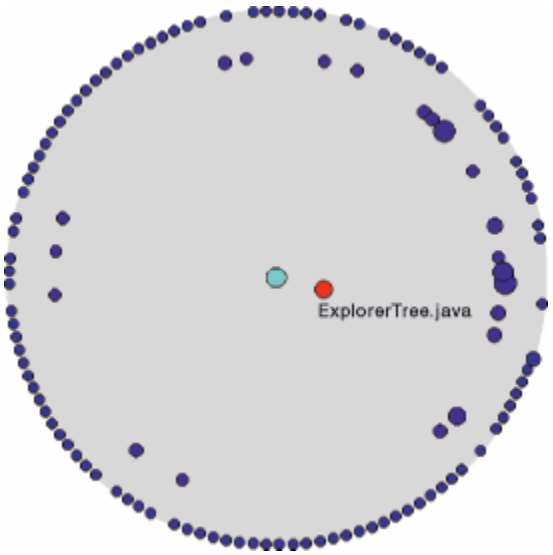
Evolution Radar: example (cnt'd)

- **Why did the CC of File*.java increase?**
- **Make a new "module in focus" from these three files and check which file of Explorer is closest**



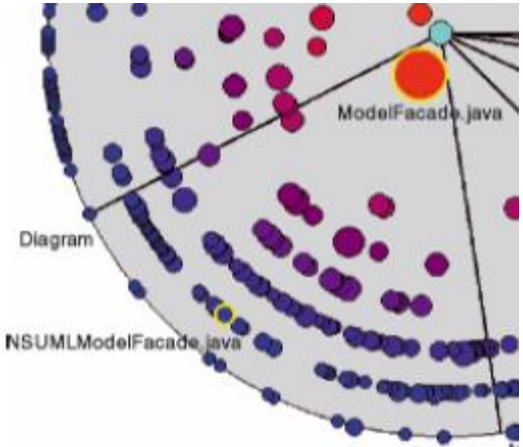
Evolution Radar: example (cnt'd)

- **Why did the CC of File*.java increase?**
- **Make a new "module in focus" from these three files and check which file of Explorer is closest**

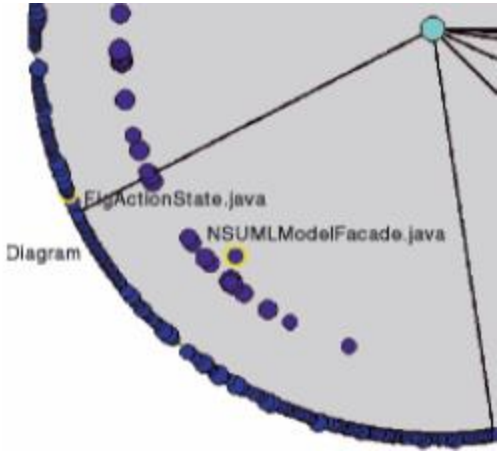


- **Problematic file was copied and removed**

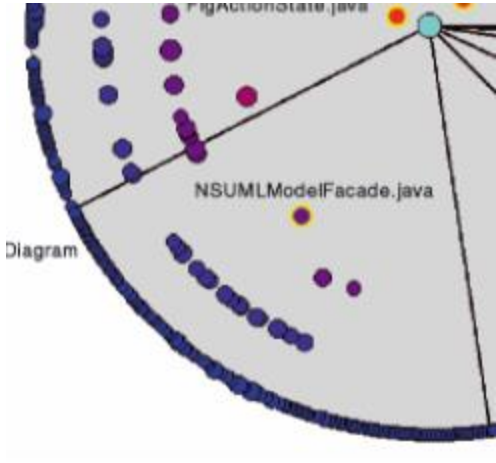
Jun-Dec 2004



Jan-Jun 2004

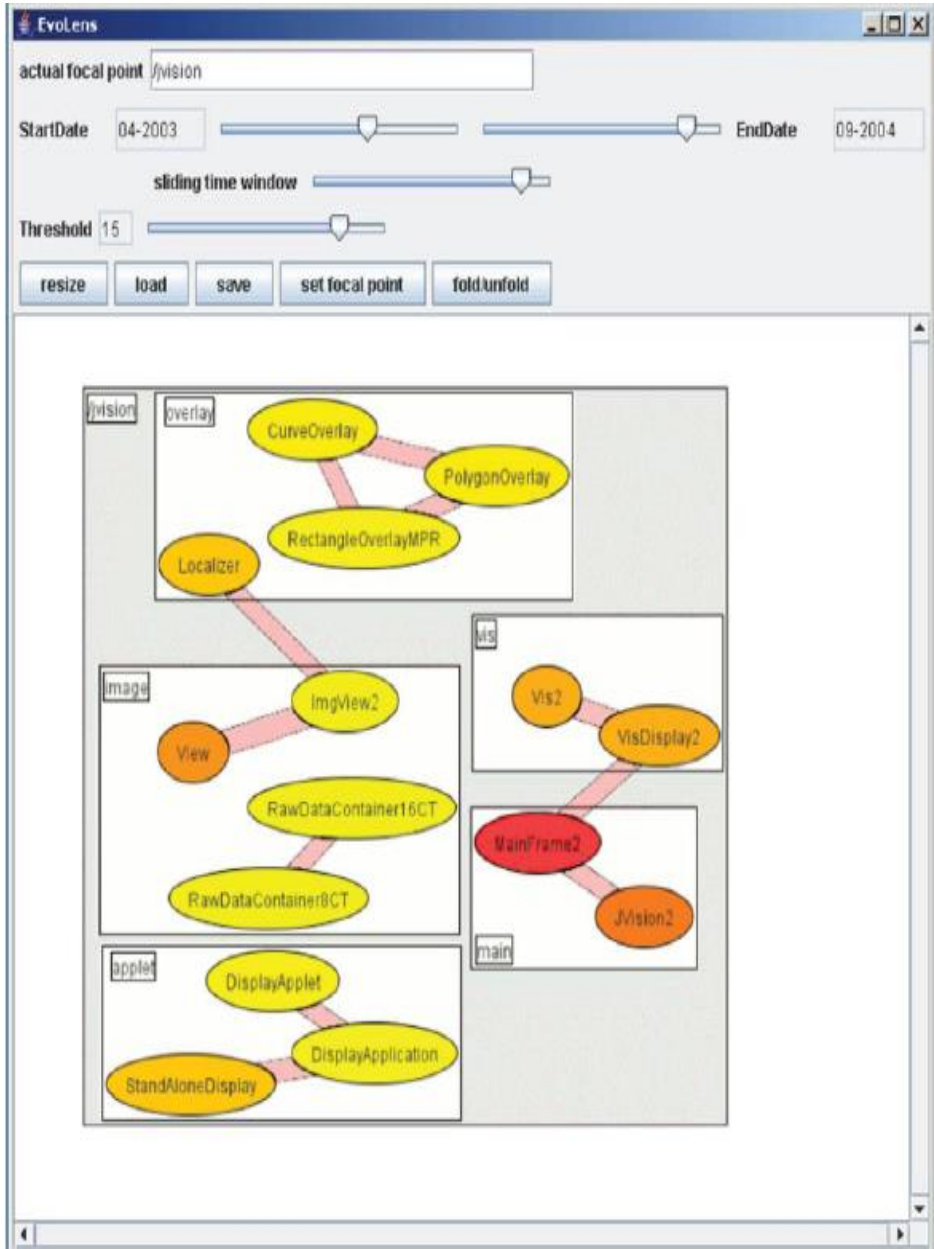


Jun-Dec 2005



Alternative visualization: EvoLens

- **Focus: gray rectangle**
- **2 hierarchy levels: classes are “flattened” to submodules**
- **Colours: growth speed**
- **Edges: strength of change coupling**
- **[Ratzinger, Fischer, Gall, 2005]**

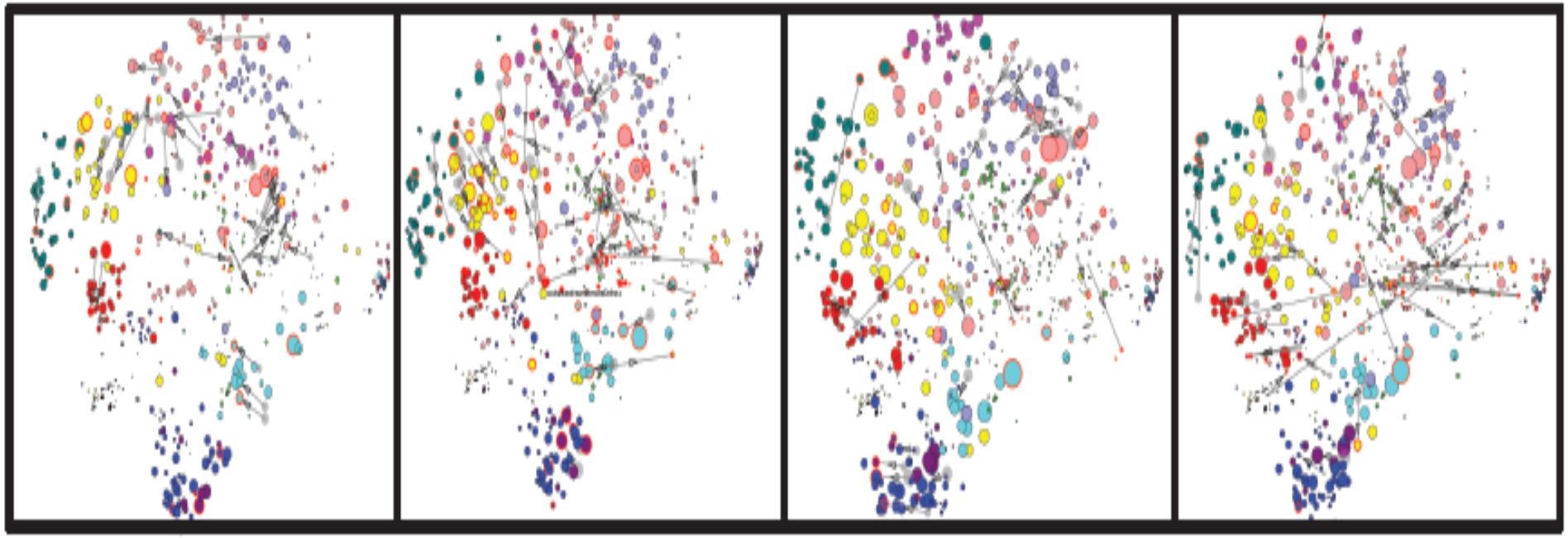


Dependencies + changes [Beyer Hassan 2006]

- "Dependency graph in time"

- Distance between the spots – change coupling
- Colours – subsystems
  - Gray and Arrow: previous version of…
- Size – #nodes the node depends upon
- Red ring = "new size" – "old size"
  - 0, if the result < 0

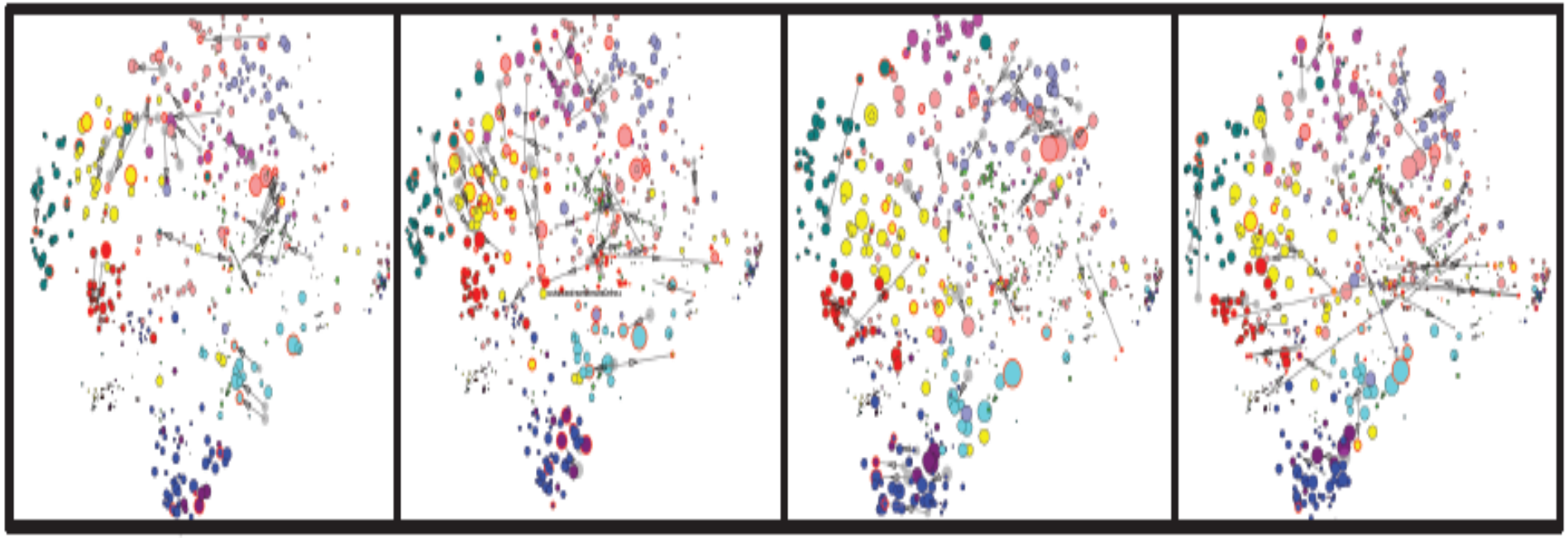
Storyboard: POSTGRESQL



2001-07-01 to 2001-10-01 | 2002-10-01 to 2003-01-01 | 2003-10-01 to 2004-01-01 | 2005-04-01 to 2005-07-01

- **What can we learn from this storyboard?**

Storyboard: POSTGRESQL



2001-07-01 to 2001-10-01 | 2002-10-01 to 2003-01-01 | 2003-10-01 to 2004-01-01 | 2005-04-01 to 2005-07-01

- **What can we learn from this storyboard?**
  - **Red (Executor) and Blue (Optimizer) are moving closer**
    - **Likely to become more dependent on each other**
  - **Yellow (Query Evaluation Engine) moves a lot**

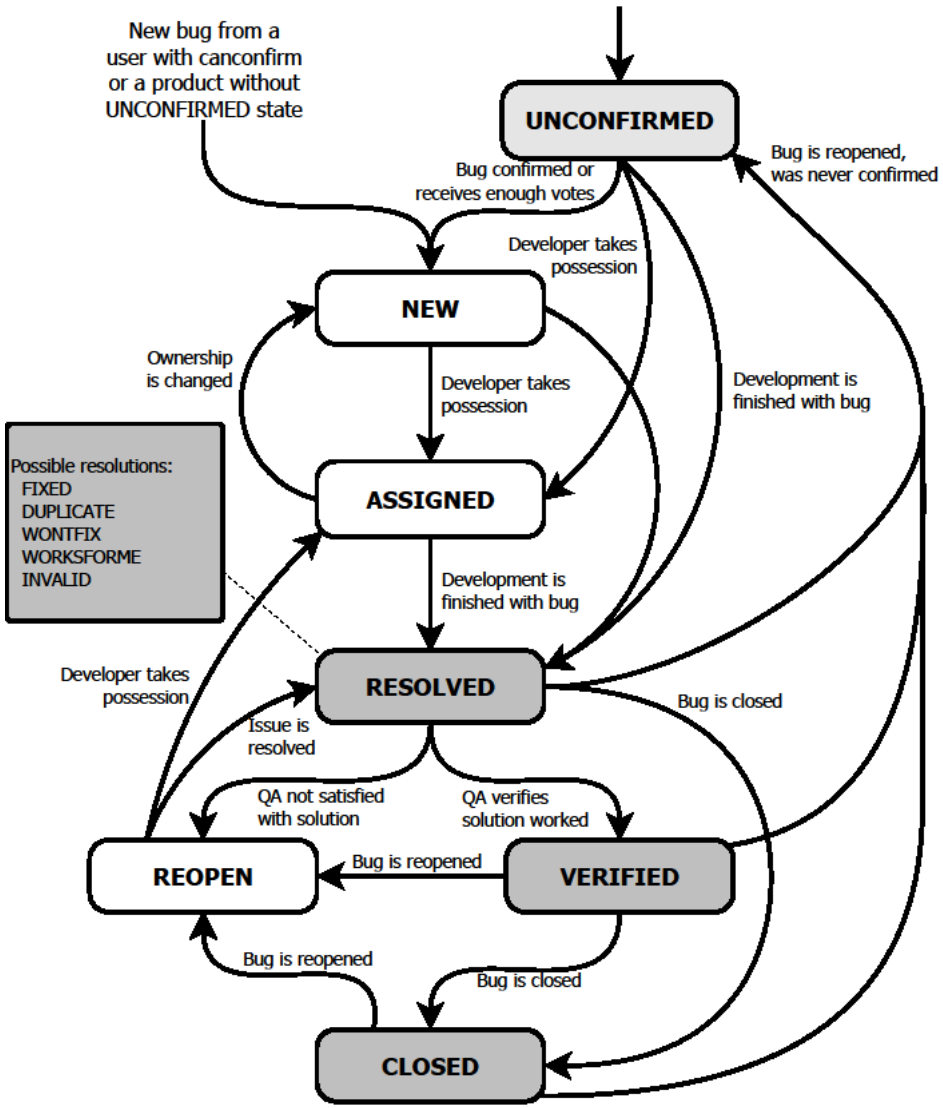
Learning about files: summary

- **Change coupling - two artifacts change together**
  - **Correlates with the number of bugs**
  - **Used to analyse relations between the files**
    - **Evolution Radar, EvoLens**
  - **Can be used in combination with dependencies**
    - **Evolution Storyboards**

Learning about bugs…

- How is the bug repository used?
- When are the bugs introduced?
- Can we predict bugs?
- Who should fix the bugs?

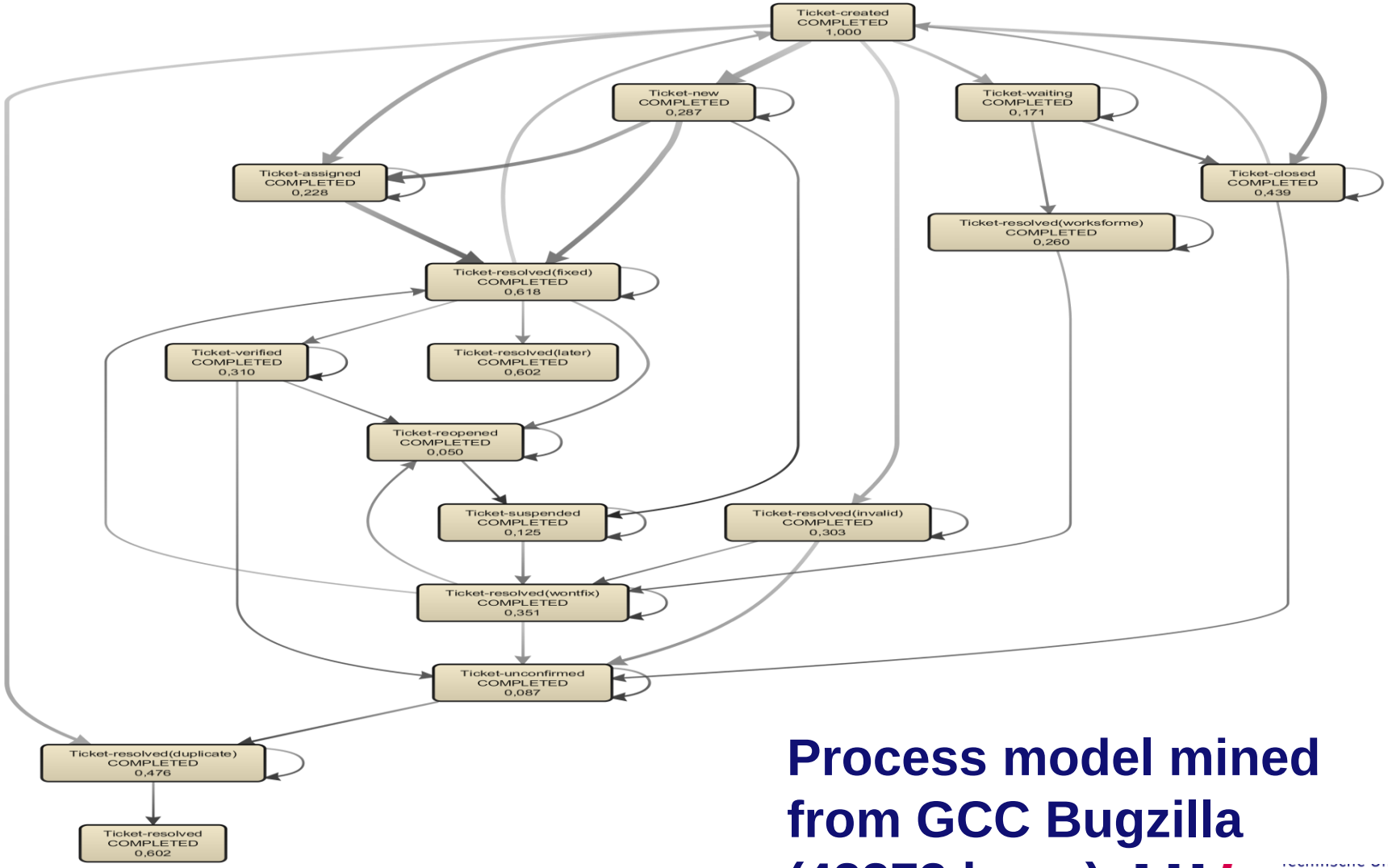
How is Bugzilla used?



"Theory": Bugzilla Guide

Practice (GCC)

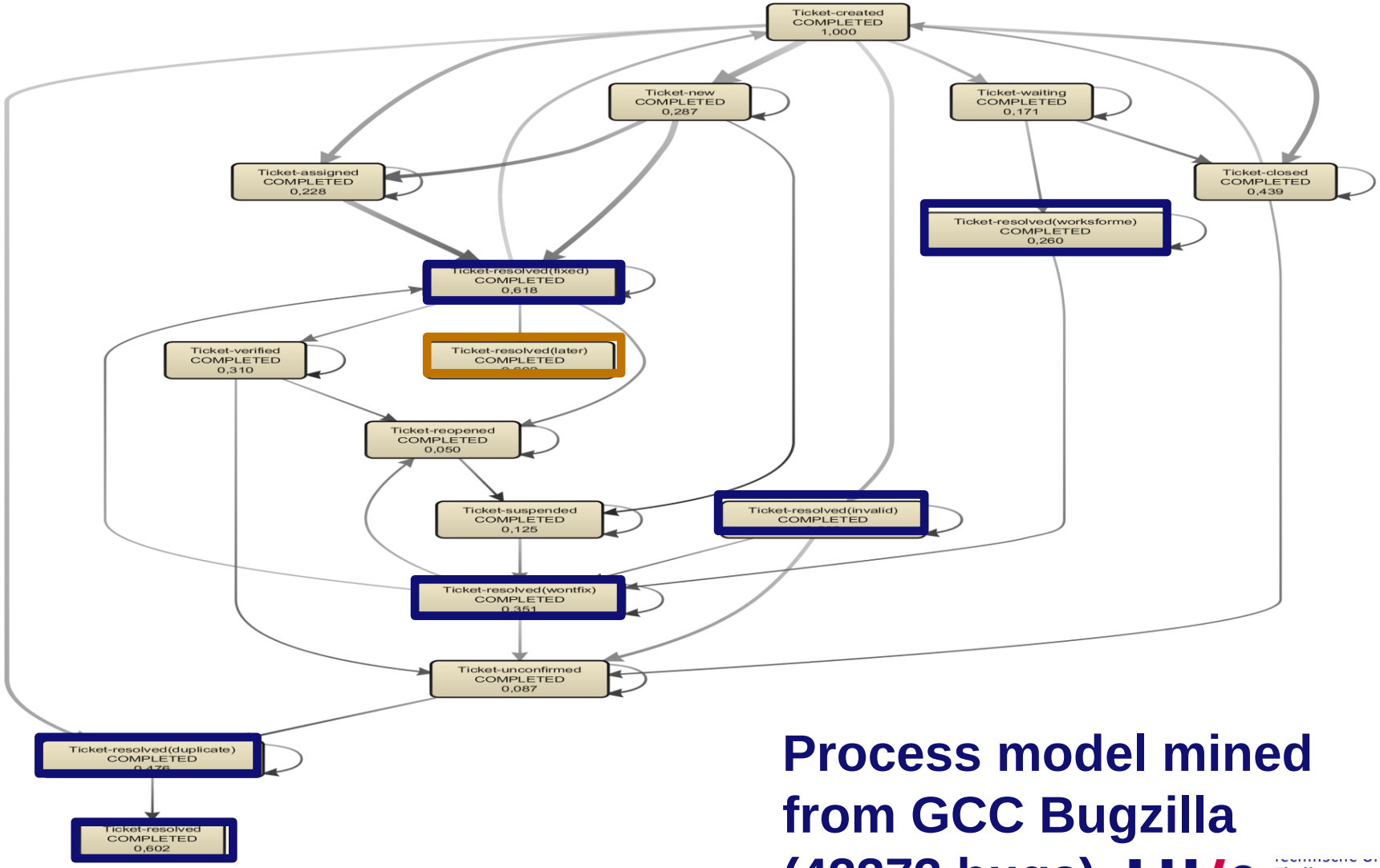
Poncin, Serebrenik, vd Brand 2011



Process model mined from GCC Bugzilla (42373 bugs)

Practice (GCC)

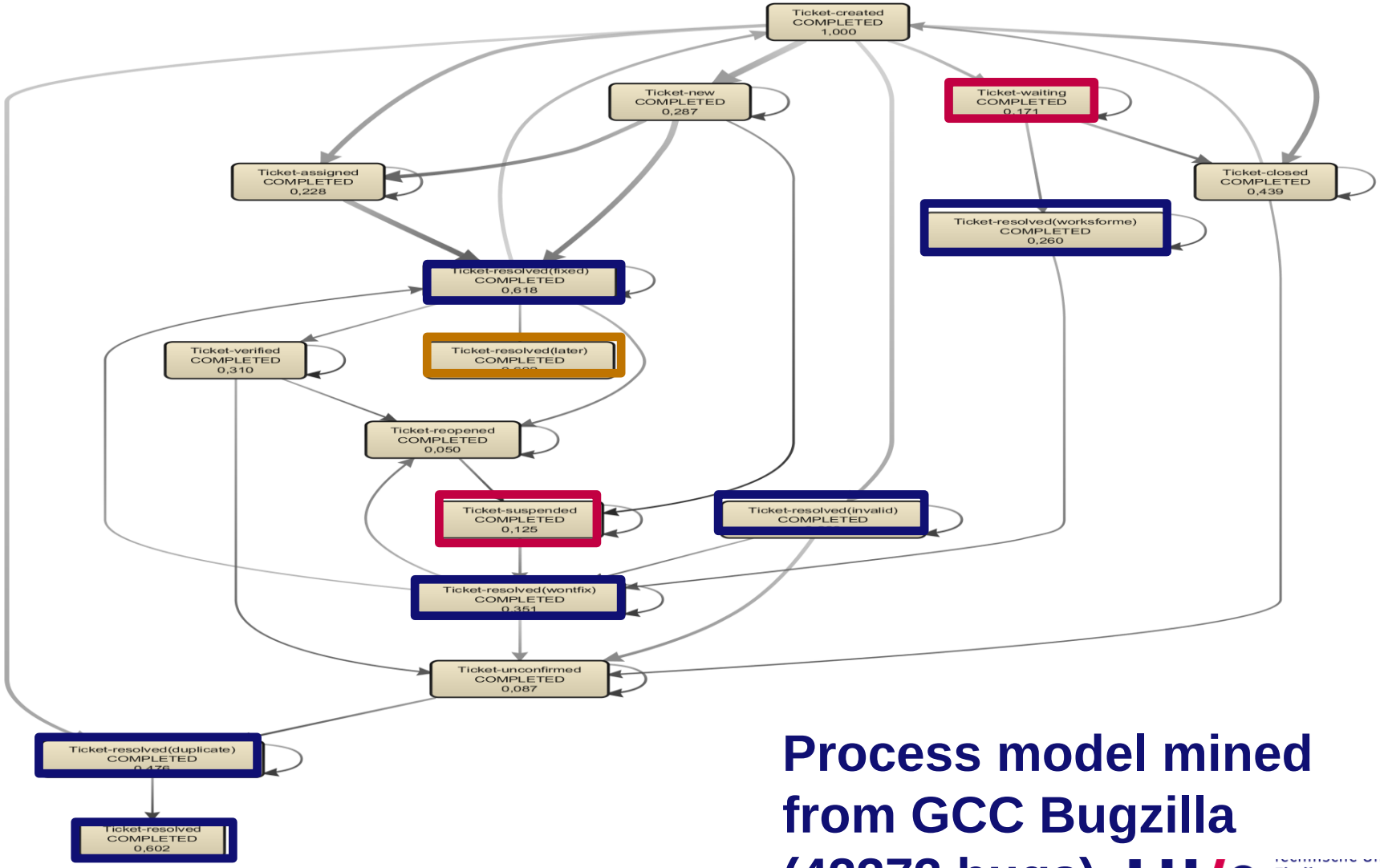
Poncin, Serebrenik, vd Brand 2011



Process model mined from GCC Bugzilla (42373 bugs)

Practice (GCC)

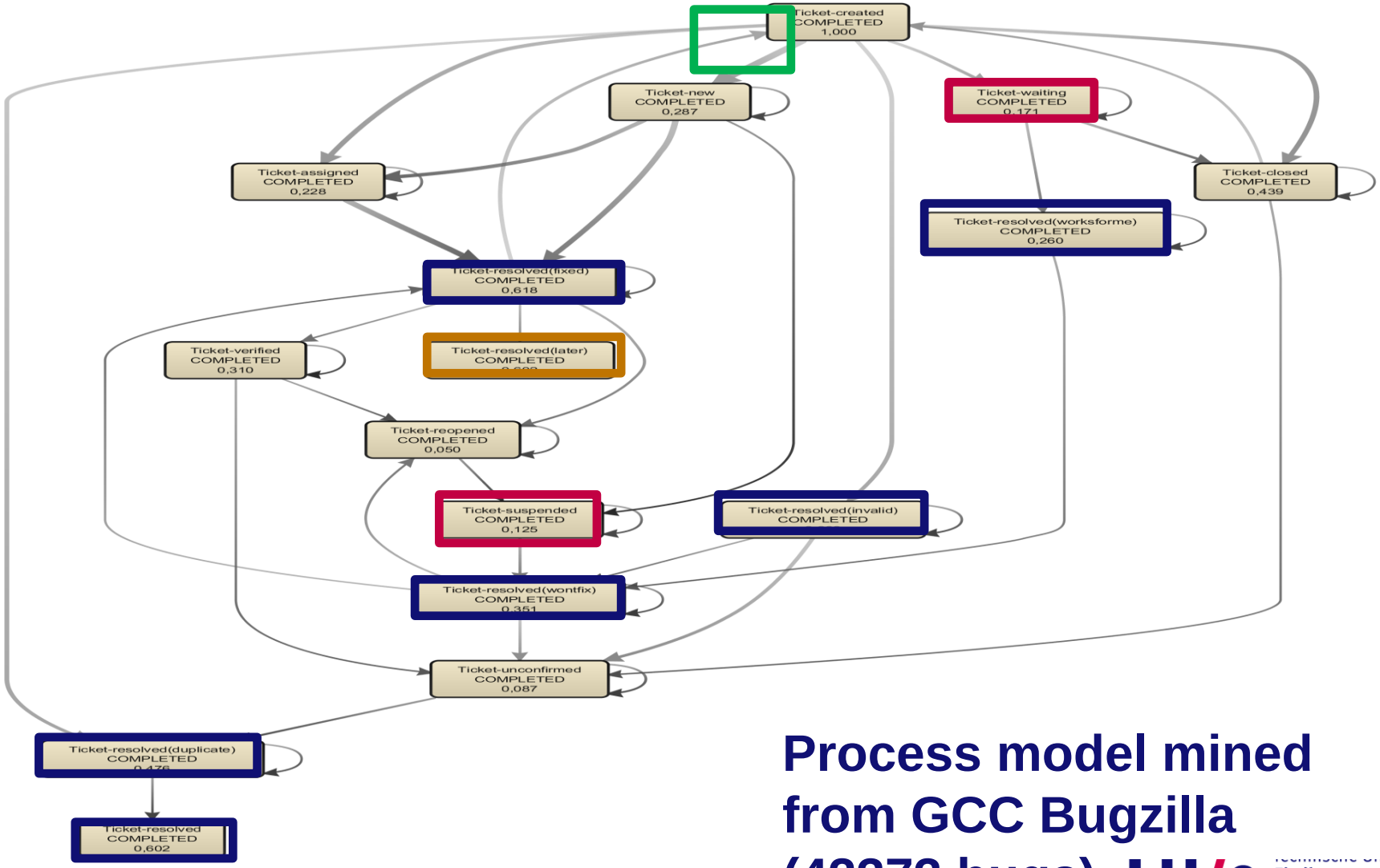
Poncin, Serebrenik, vd Brand 2011



Process model mined from GCC Bugzilla (42373 bugs)

Practice (GCC)

Poncin, Serebrenik, vd Brand 2011



Process model mined from GCC Bugzilla (42373 bugs)

Bugs and Source code

Bug 22554 - [4.1 Regression] pb_assoc header build and install overflows exec

Status: RESOLVED FIXED

Product: gcc
Component: libstdc++
Version: 4.1.0

Importance: P2 normal
Target Milestone: 4.1.0
Assigned To: Benjamin Kosnik

URL:
Keywords: build

Depends on:
Blocks: ~~23734~~
Show dependency tree / graph

Reported

Reported: 2005-07-19 04:19 UTC by David Edelsohn
Modified: ~~2005-09-15 18:59 UTC (History)~~
CC List: 2 users (show)

See Also:
Host: powerpc-ibm-aix
Target: powerpc-ibm-aix
Build: powerpc-ibm-aix
Known to work:
Known to fail:
Last reconfirmed: 2005-07-19 04:23:46

Attachments	
Makefile workaround (8.44 KB, patch) 2005-07-19 04:22 UTC, David Edelsohn	Details | Diff
fix (8.95 KB, patch) 2005-09-12 22:48 UTC, Benjamin Kosnik	Details | Diff
Add an attachment (proposed patch, testcase, etc.) View All	

Resolved

But when was the bug introduced?

Which files did one change when fixing the bug?

[patch] fix

p.20050912-3 (text/plain), 8.95 KB, created by Benjamin Ko

```
2005-09-12  Benjamin Kosnik  <bkoz@
            David Edelsohn <dje@gcc

        PR libstdc++/22554
        PR libstdc++/23734
        * include/Makefile.am (asso
        (install-headers): Use them
        (stamp-assoc): Same.
        * include/Makefile.in: Rege

Index: include/Makefile.am
===================================
RCS file: /cvs/gcc/gcc/libstdc++-v3/
retrieving revision 1.105
diff -c -p -r1.105 Makefile.am
*** include/Makefile.am 17 Aug 2005
--- include/Makefile.am 12 Sep 2005
```

projects / gcc.git / commit

summary | shortlog | log | commit | commitdiff | tree
(parent: 6cab81b) | patch

2005-09-12 David Edelsohn <dje@gcc.gnu.org>

```
author     bkoz <bkoz@138bc75d-0d04-0410-961f-82ee72b054a4>
           Tue, 13 Sep 2005 19:22:52 +0000 (19:22 +0000)
committer  bkoz <bkoz@138bc75d-0d04-0410-961f-82ee72b054a4>
           Tue, 13 Sep 2005 19:22:52 +0000 (19:22 +0000)
commit     1fb41e7061f2f03af9bd94d07c74a1260da2f949
tree       f548dd3737f17a69909461a772f76a25e19459c3        tree | snapshot
parent     6cab81bd360425f959c01e66ba5c29f7adde5139        commit | diff
```

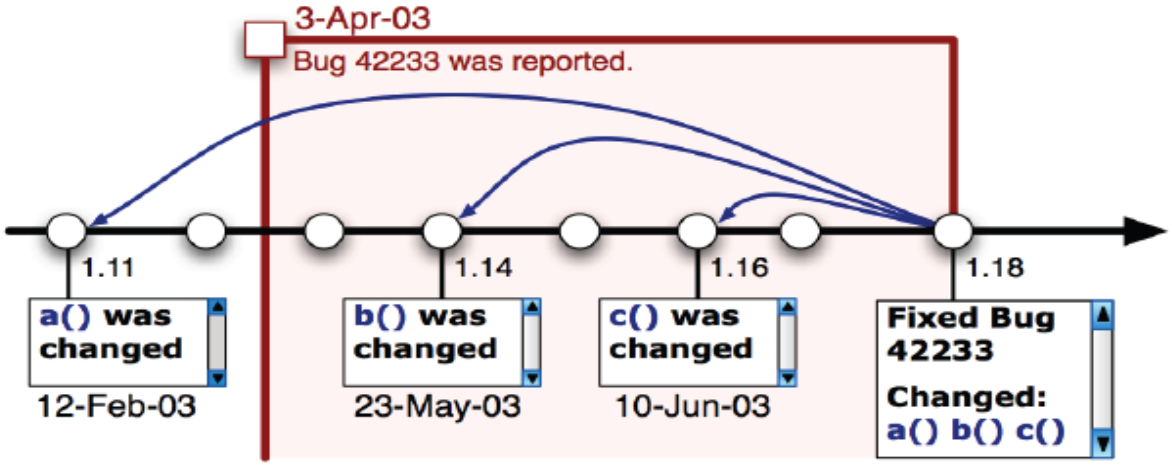
```
2005-09-12  David Edelsohn  <dje@gcc.gnu.org>

PR libstdc++/22554
PR libstdc++/23734
* include/Makefile.am (stamp-assoc): Install each subgroup
of headers separately.
* include/Makefile.in: Regenerate.

git-svn-id: svn+ssh://gcc.gnu.org/svn/gcc/trunk@104238 138bc75d-0d04-0410-961f-82ee72b054a4
```

- **Heuristic**: If a bug is fixed, it will be documented in the commit (#22554)
  - Time stamps, author names, words like “fix” or “bug”
- **Commercial solutions provide better linking**

When was the bug introduced?



With gratitude to Marco D'Ambros and Romain Robbes

- **Revisions 1.14 and 1.16 could not have introduced the bug!**
- **Ignore commits that**
  - **Affect only comments and whitespaces**
  - **Affect numerous files at once (license changes, merges)**

Surprising result…

When Do Changes Induce Fixes?

(On Fridays.)

Jacek Śliwerski
International Max Planck Research School
Max Planck Institute for Computer Science
Saarbrücken, Germany
sliwers@mpi-sb.mpg.de

Thomas Zimmermann Andreas Zeller
Department of Computer Science
Saarland University
Saarbrücken, Germany
{tz, zeller}@acm.org

Back to the future

- **Once we know which files have introduced bugs in the past, we try to predict which files will introduce bugs in the future**
  - **Metrics – more in the lectures to come**
  - **History – today!**

- **Very important research domain: defect prediction**

Predicting defects with BugCache

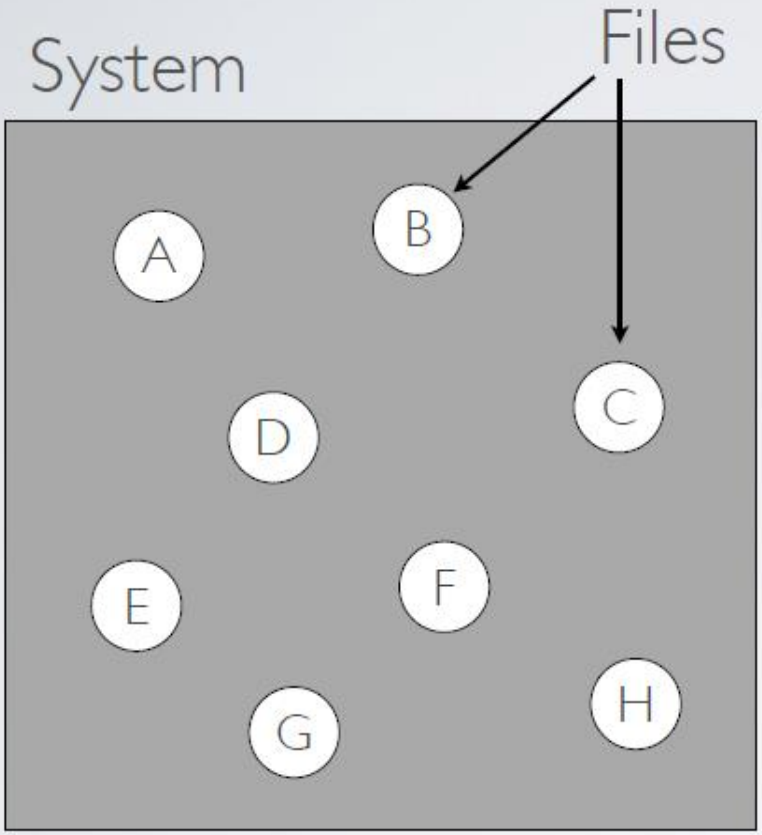
- Online prediction approach based on 5 principles:

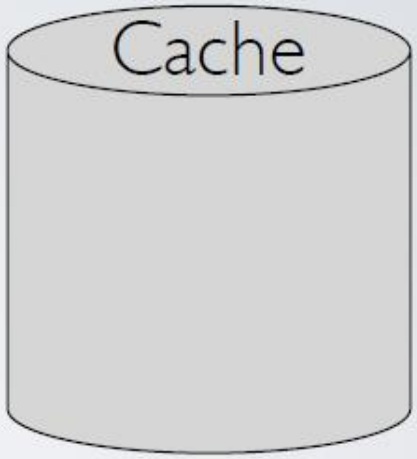
  1. Spatial locality: bug-prone artifacts are changed together
  2. Temporal locality: artifacts which exhibit bugs in the past are likely to exhibit them also in the future
  3. Bugs are found in new artifacts
  4. Bugs are found in frequently modified artifacts
  5. Bugs are found in large artifacts

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

With gratitude to Marco D'Ambros and Romain Robbes

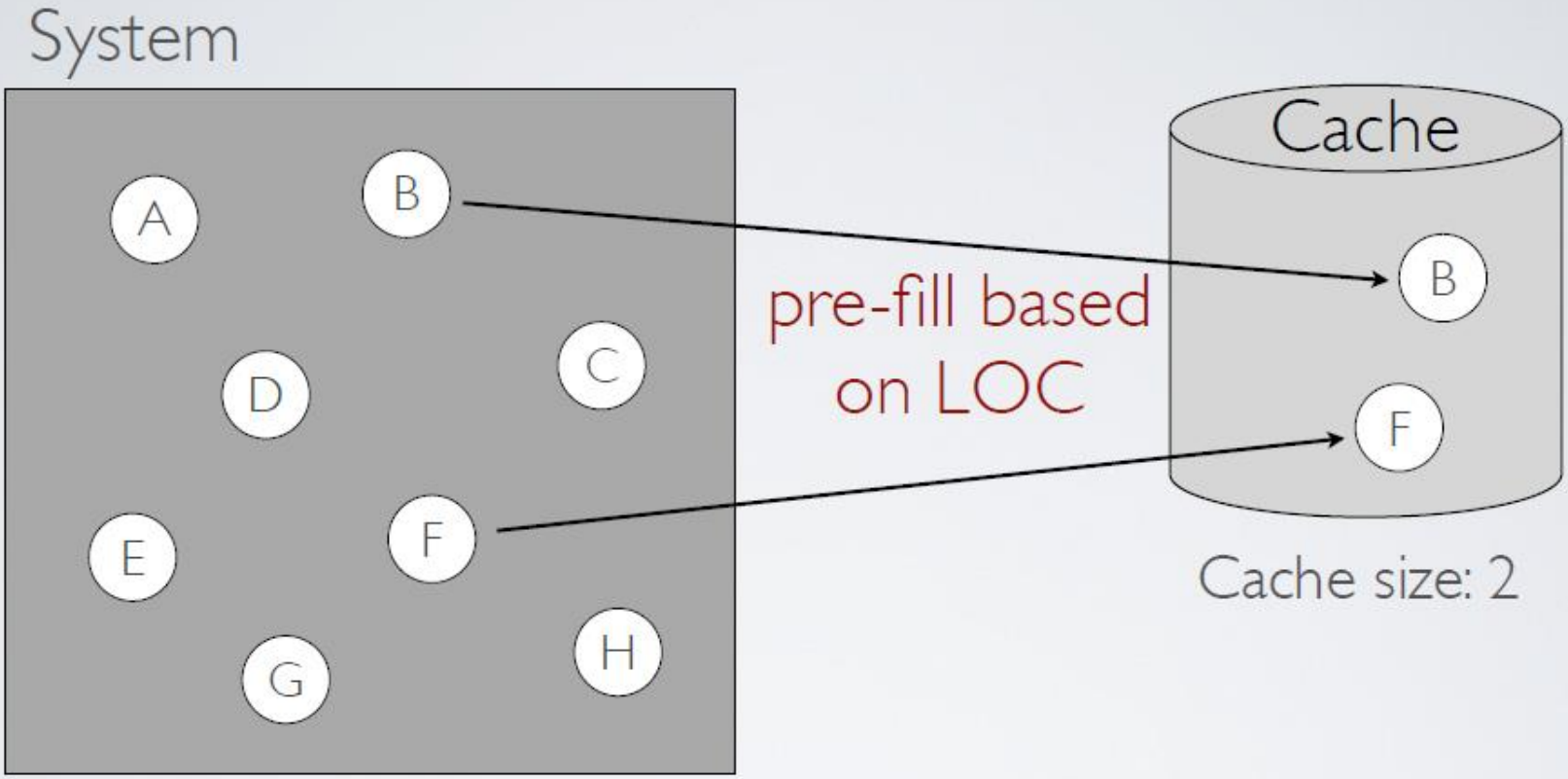
Predicting defects with BugCache



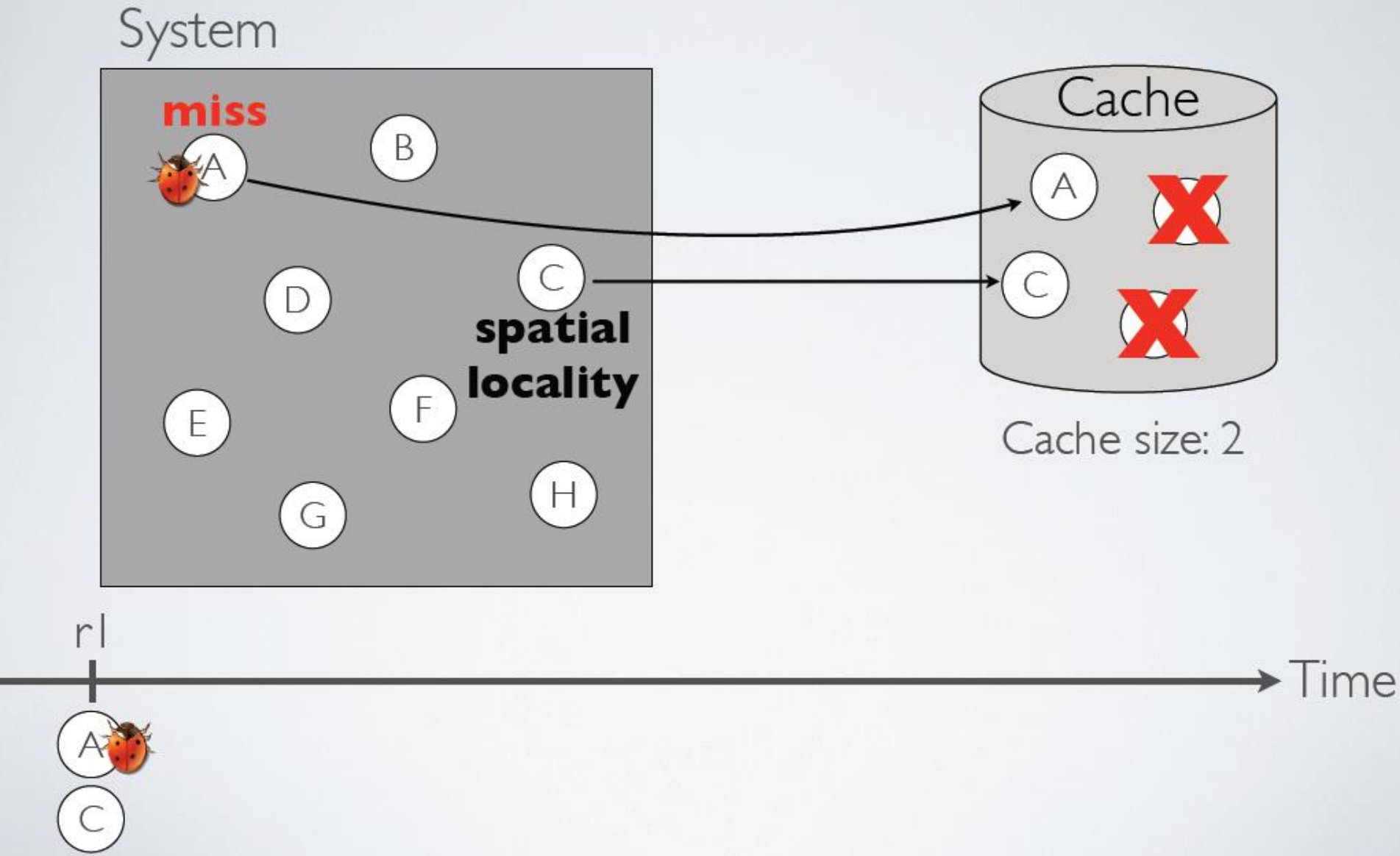


With gratitude to Marco D'Ambros and Romain Robbes

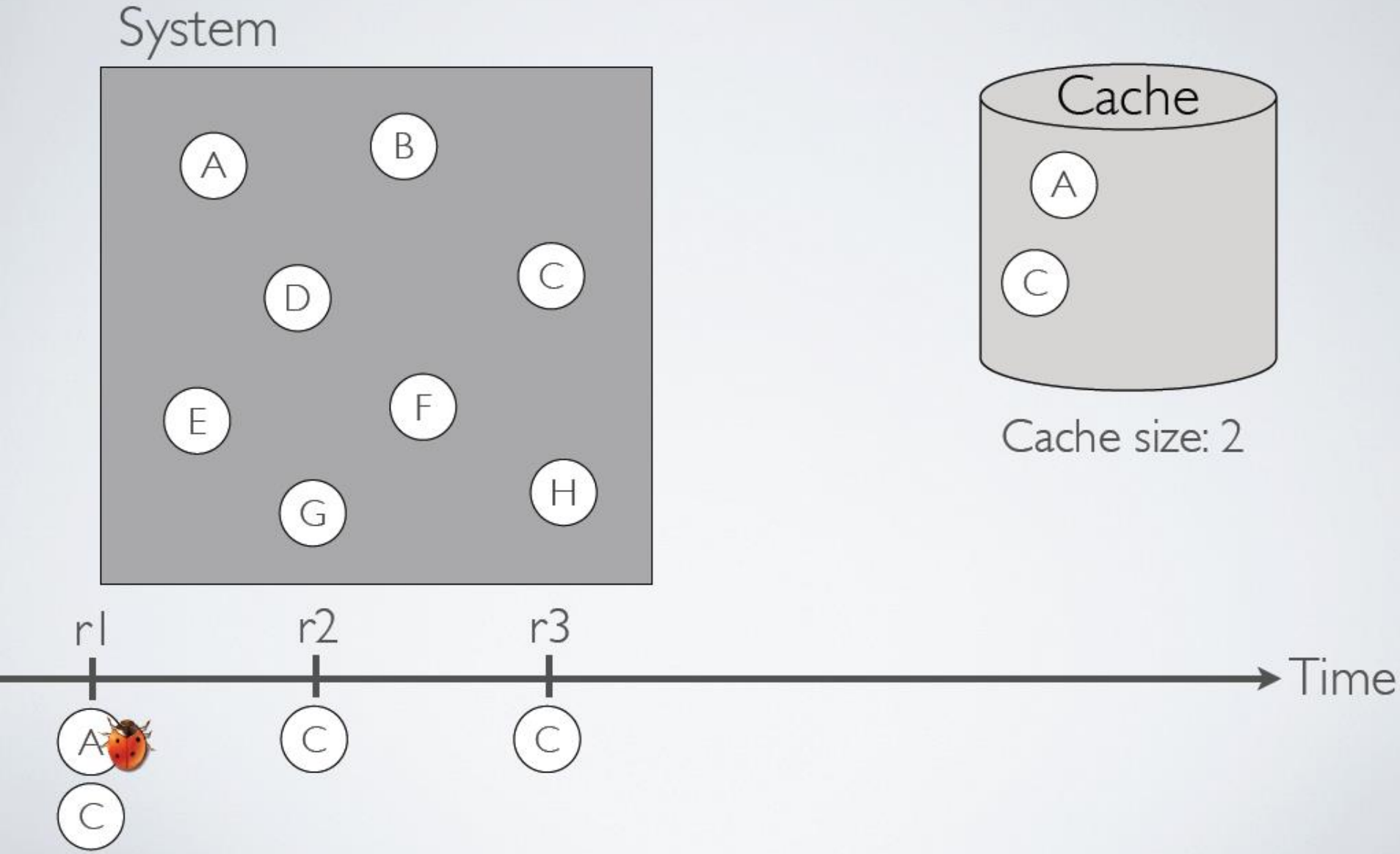
Predicting defects with BugCache



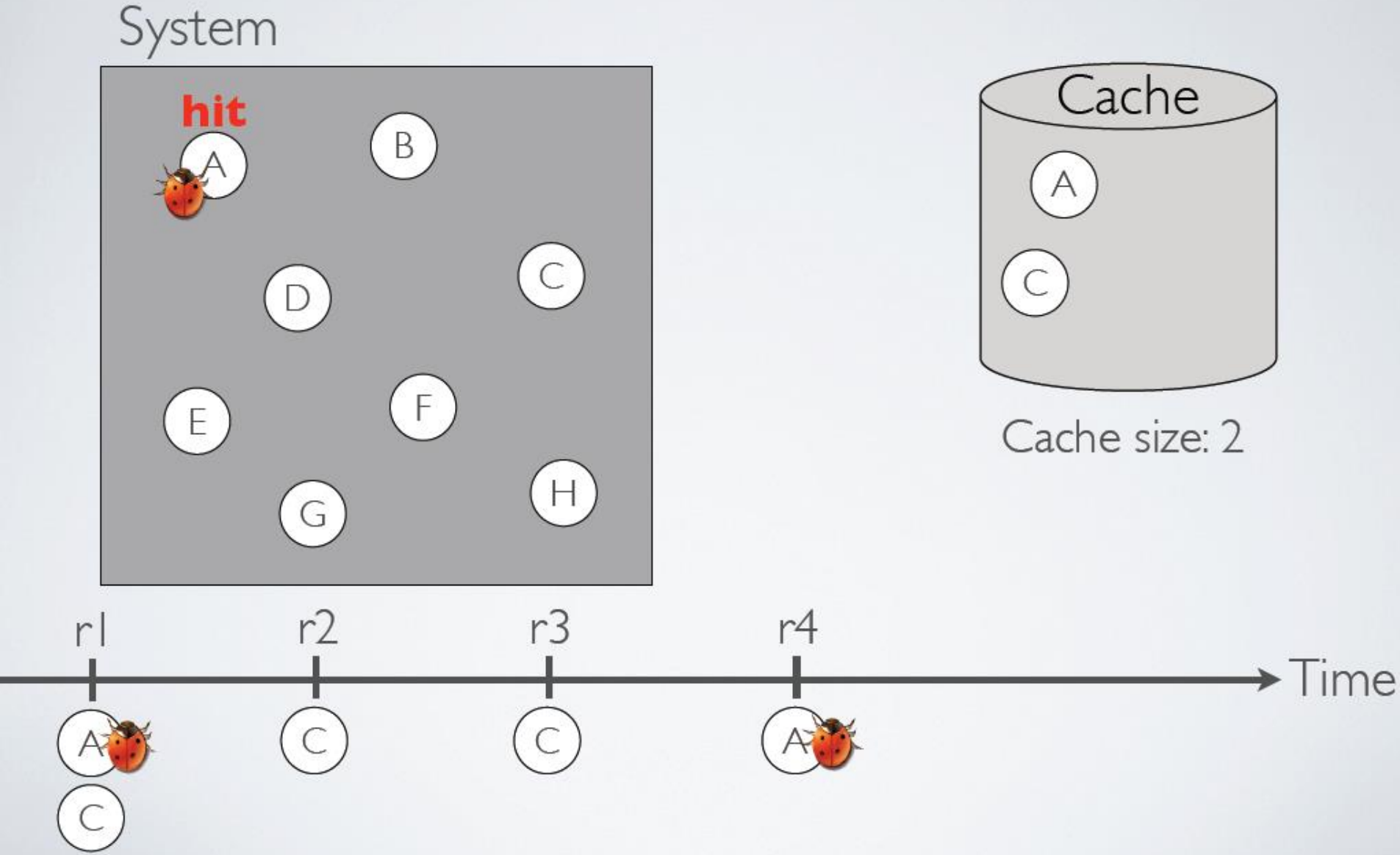
Predicting defects with BugCache



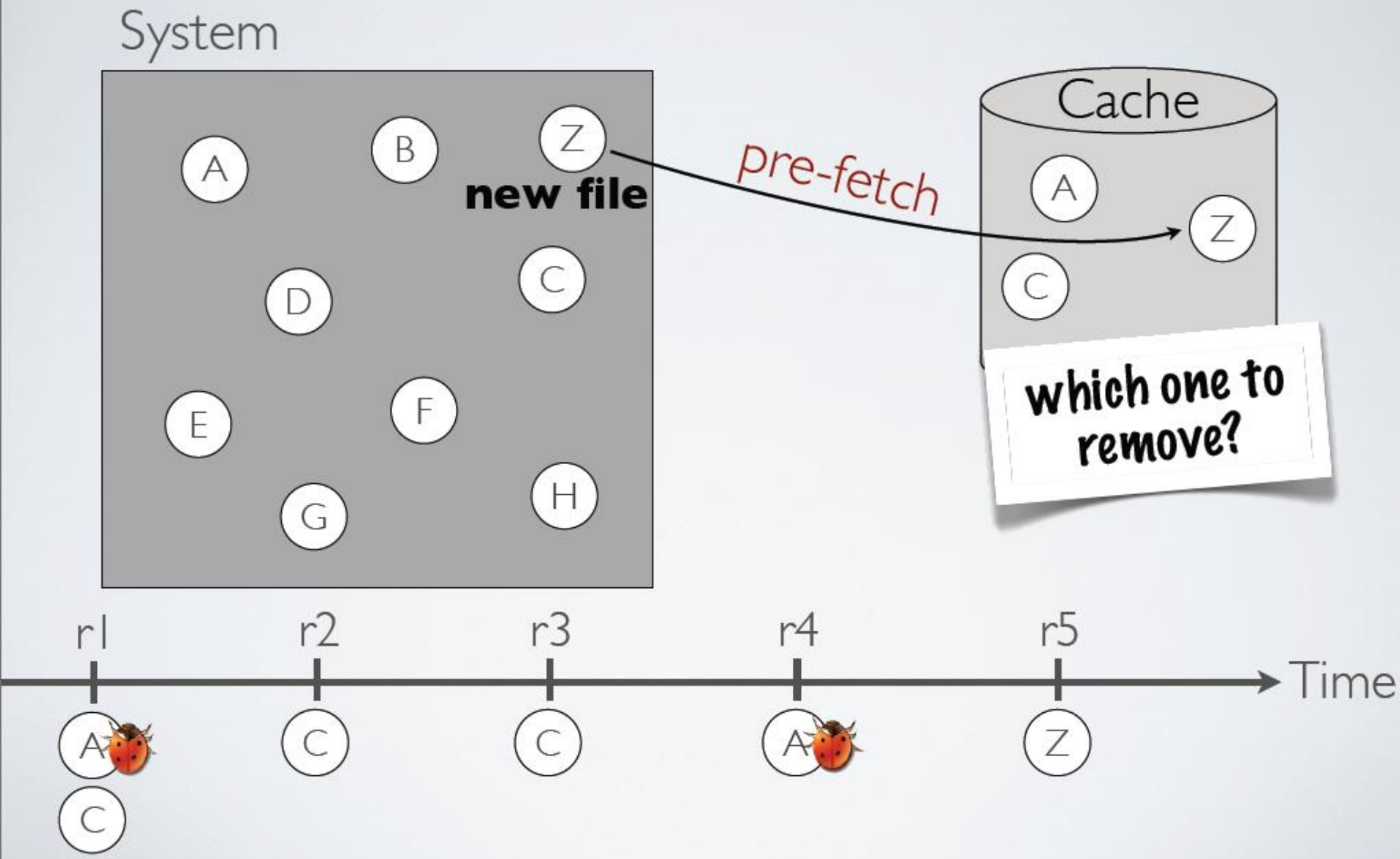
Predicting defects with BugCache



Predicting defects with BugCache

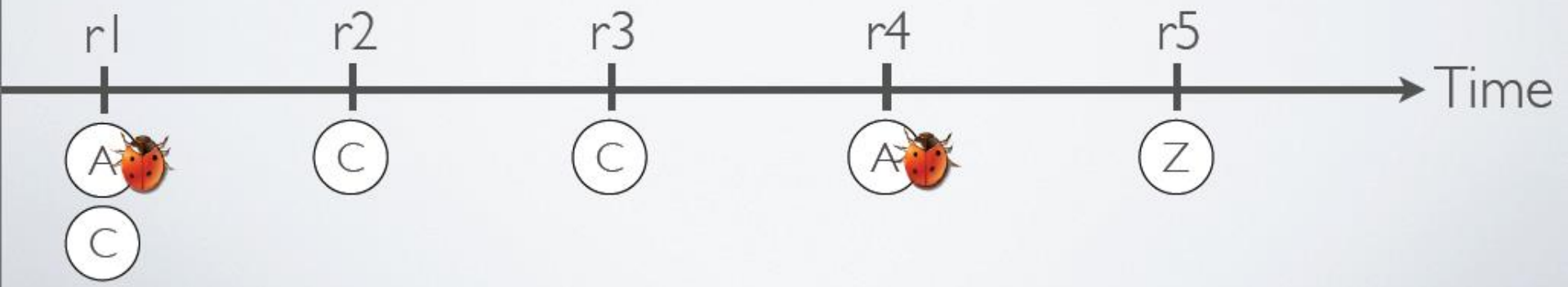


Predicting defects with BugCache



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

With gratitude to Marco D'Ambros and Romain Robbes

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%

With gratitude to Marco D'Ambros and Romain Robbes

However…

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

With gratitude to Marco D'Ambros and Romain Robbes

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

With gratitude to Marco D'Ambros and Romain Robbes

From introduction back to resolution

- **Who should resolve the bugs?**

- **Someone who had similar bugs in the past!** [Anvik et al. 2006]
  - **Preprocess existing bug reports:** remove the stop-words, calculate frequencies of the terms…
  - **Identify the developer that fixed the bug**: “assigned-to” might be empty or incorrect
  - **Ignore bug reports** fixed by developers that left the project
  - Train a **machine learning** algorithm to classify bug reports

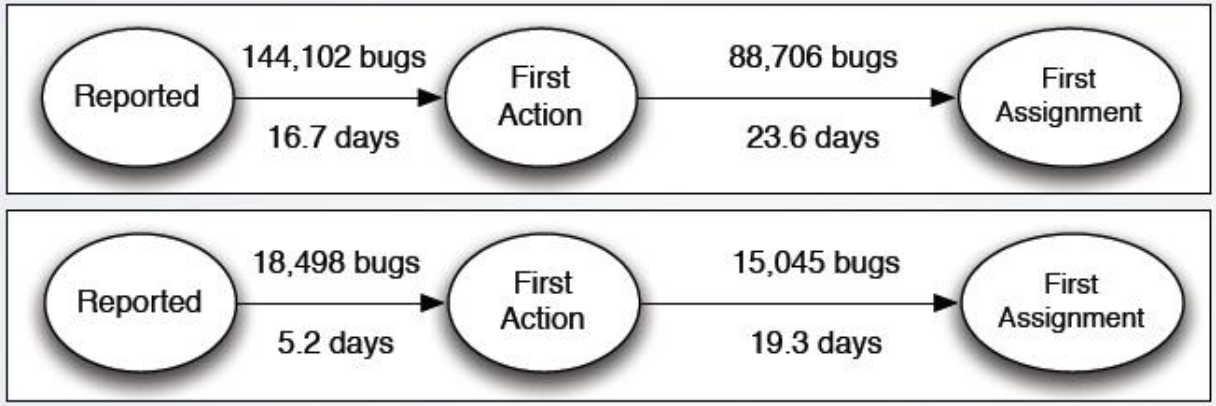
Triaging results

- 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

With gratitude to Marco D'Ambros and Romain Robbes

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

With gratitude to Marco D'Ambros and Romain Robbes

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

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

With gratitude to Marco D'Ambros and Romain Robbes

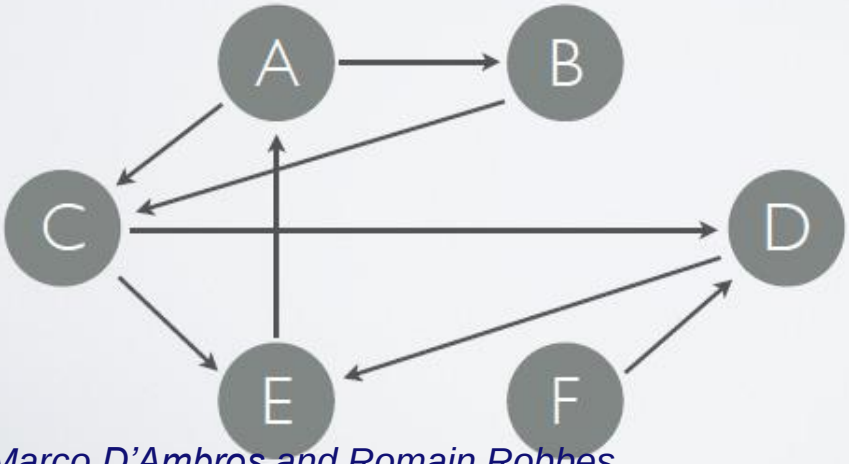
Modeling tossing graph with Markov chain

- 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

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



With gratitude to Marco D'Ambros and Romain Robbes

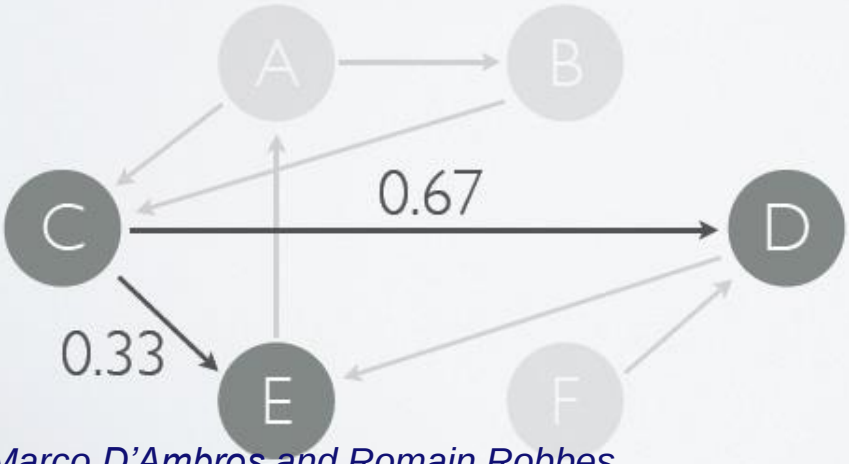
Modeling tossing graph with Markov chain

- 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

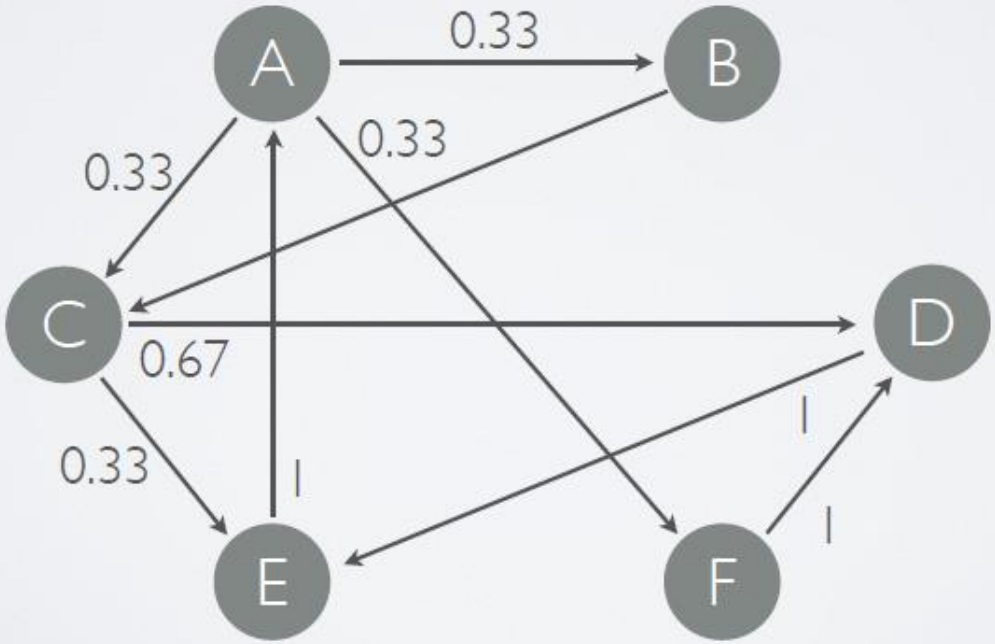
$A \to B\,(1),\ B \to C\,(1),$
$C \to D\,(2),\ A \to C\,(1),$
$D \to E\,(1),\ C \to E\,(1),$
$E \to A\,(1),\ A \to F\,(1),$
$F \to D\,(1)$



With gratitude to Marco D'Ambros and Romain Robbes

Minimum support and transaction probability

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

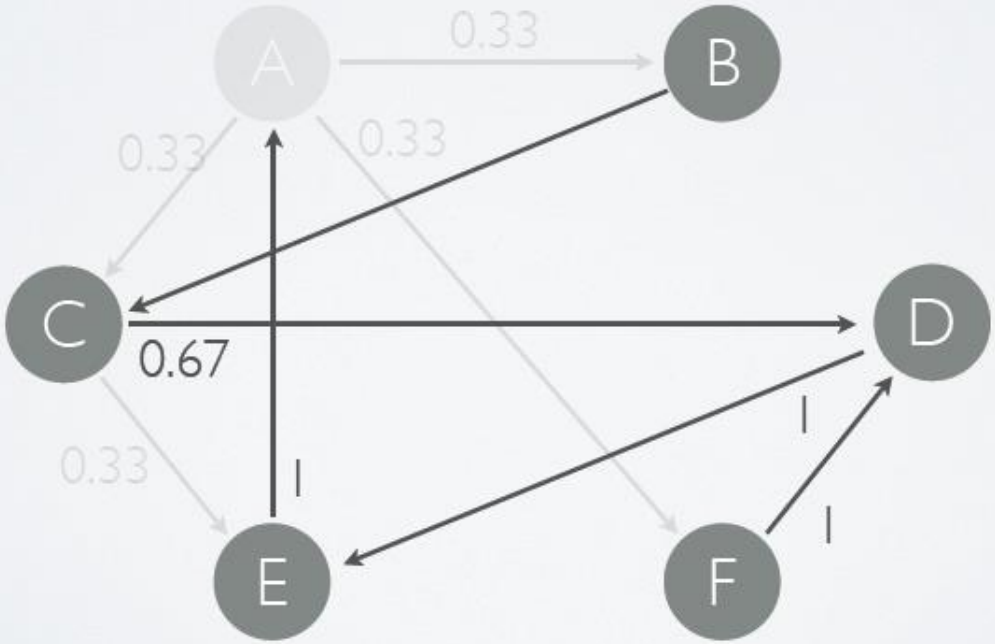


With gratitude to Marco D'Ambros and Romain Robbes

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

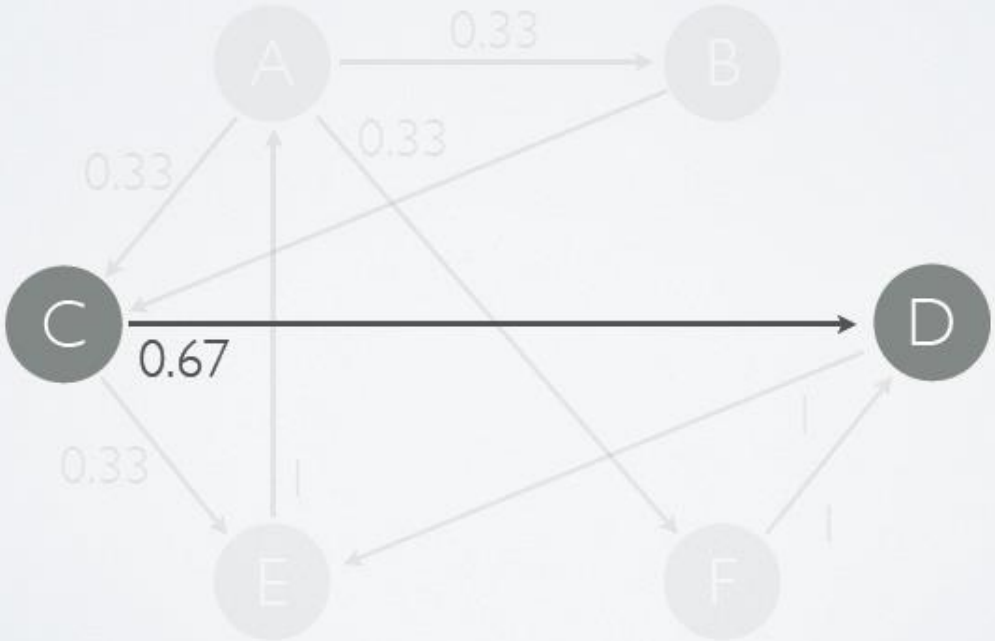


With gratitude to Marco D'Ambros and Romain Robbes

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



With gratitude to Marco D'Ambros and Romain Robbes

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

With gratitude to Marco D'Ambros and Romain Robbes

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	[illegible]
		first 4	52[illegible]	[illegible]	[illegible]
		first 5	59[illegible]	[illegible]	[illegible]
	Bayesian Network	first 2	40[illegible]	[illegible]	[illegible]
		first 3	50[illegible]	[illegible]	[illegible]
		first 4	55.[illegible]	[illegible]	12.05
		first 5	59.53	70.82	11.29

Improvement up to 23%
Accuracy up to 71%

Not all bugs are equally interesting…

- **Effort / priority**
- **Duplicate bugs**
- **Related bugs**
  - **E.g., the same solution for different bugs**

EINDHOVEN UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

MASTER'S THESIS

Related Defect Report Detection using an Information Retrieval Approach

October 4, 2013

Author: ing. Y.M. Schroot
y.m.schroot@student.tue.nl

Conclusions

- **Looking at the version control systems' logs we can learn about files and bugs**

- **We can even predict the future!**
  - **To a certain extent…**

2IS55 Software Evolution

Software metrics

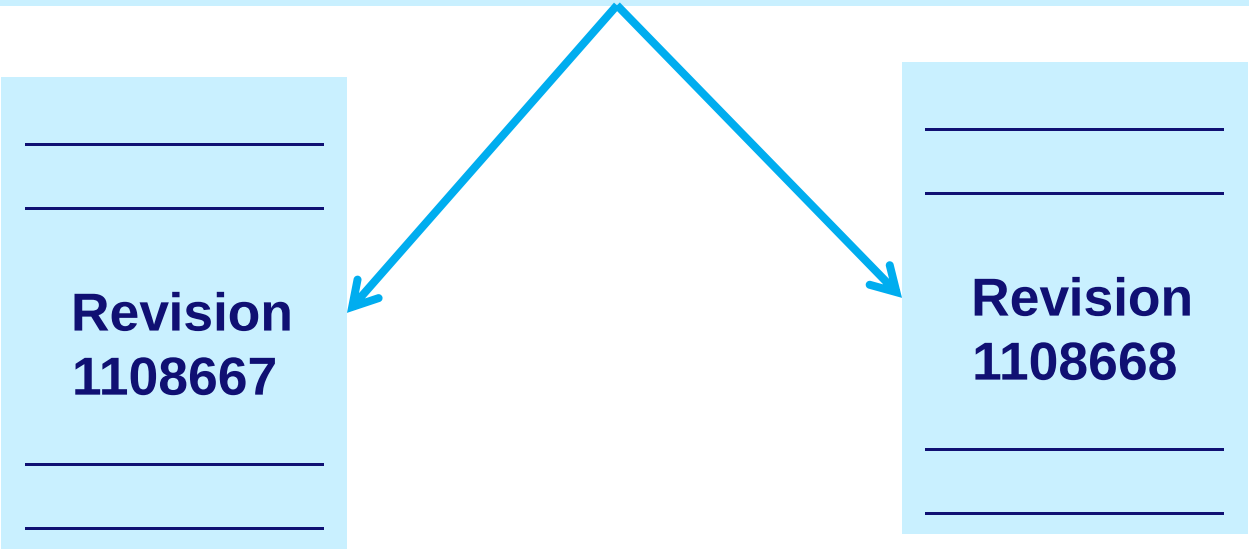
Alexander Serebrenik

Technische Universiteit Eindhoven University of Technology

Where innovation starts

Today: Version control system is not just a log…

r1108668 | tokoe | 2010-03-29 16:54:02 +0200 (ma, 29 mrt 2010)
Changed paths:
M /trunk/KDE/kdepim/kmail/kmsearchpatternedit.cpp



- **Measure each revision**
- **Get insights in the evolution**

Why do we want to measure revisions?

- Recall the "goals-questions-<views>-metrics" approach we used for architecture reconstruction?
  - Goals: What problem does the measurement try to solve?
    - Ex.: Modifying code is experienced as difficult
    - Goal: Assess and improve maintainability of the code
  - Questions: What do we need to know to achieve the goal?
    - Is the code large? Complex? Appropriately modularized? Buggy? Documented?
  - <Views>: Which views are need to answer the questions?
    - Individual components, dependency structure
  - Metrics: How can we quantify the answers?
    - Main topic of the lecture

Measure each revision…

- **Metric:**
  - **"A quantitative measure of the degree to which a system, component, or process possesses a given variable. " --- IEEE Standard 610.12-1990**
  - **"A software metric is any type of measurement which relates to a software system, process or related documentation." --- Ian Sommerville, Software Eng. 2006**

  - **Short: mapping of software artefacts to a well-known domain**

Domains and scales

Nominal

=,≠

1-1 trans.

Imple-
mentation
language

Domains and scales

Nominal	Ordinal
=,≠ 1-1 trans. Imple-mentation language	> >-pres. trans. Priorities (high > middle > low)

Domains and scales

Nominal	Ordinal	Interval
=,≠ 1-1 trans. Imple-mentation language	> >-pres. trans. Priorities (high > middle > low)	Distance function Affine Tempe-rature (°C, °F)

Domains and scales

Nominal	Ordinal	Interval	Ratio
=,≠	>	Distance function	Zero, unit
1-1 trans.	>-pres. trans.	Affine	Linear
Imple-mentation language	Priorities (high > middle > low)	Tempe-rature (°C, °F)	m ↔ ft
			% of commits by Alice

Domains and scales

Nominal	Ordinal	Interval	Ratio	Absolute
=,≠	>	Distance function	Zero, unit	Values are absolute
1-1 trans.	>-pres. trans.	Affine	Linear	Identity
Imple-mentation language	Priorities (high > middle > low)	Tempe-rature (°C, °F)	m ↔ ft	#devel's
			% of commits by Alice	P(failure)

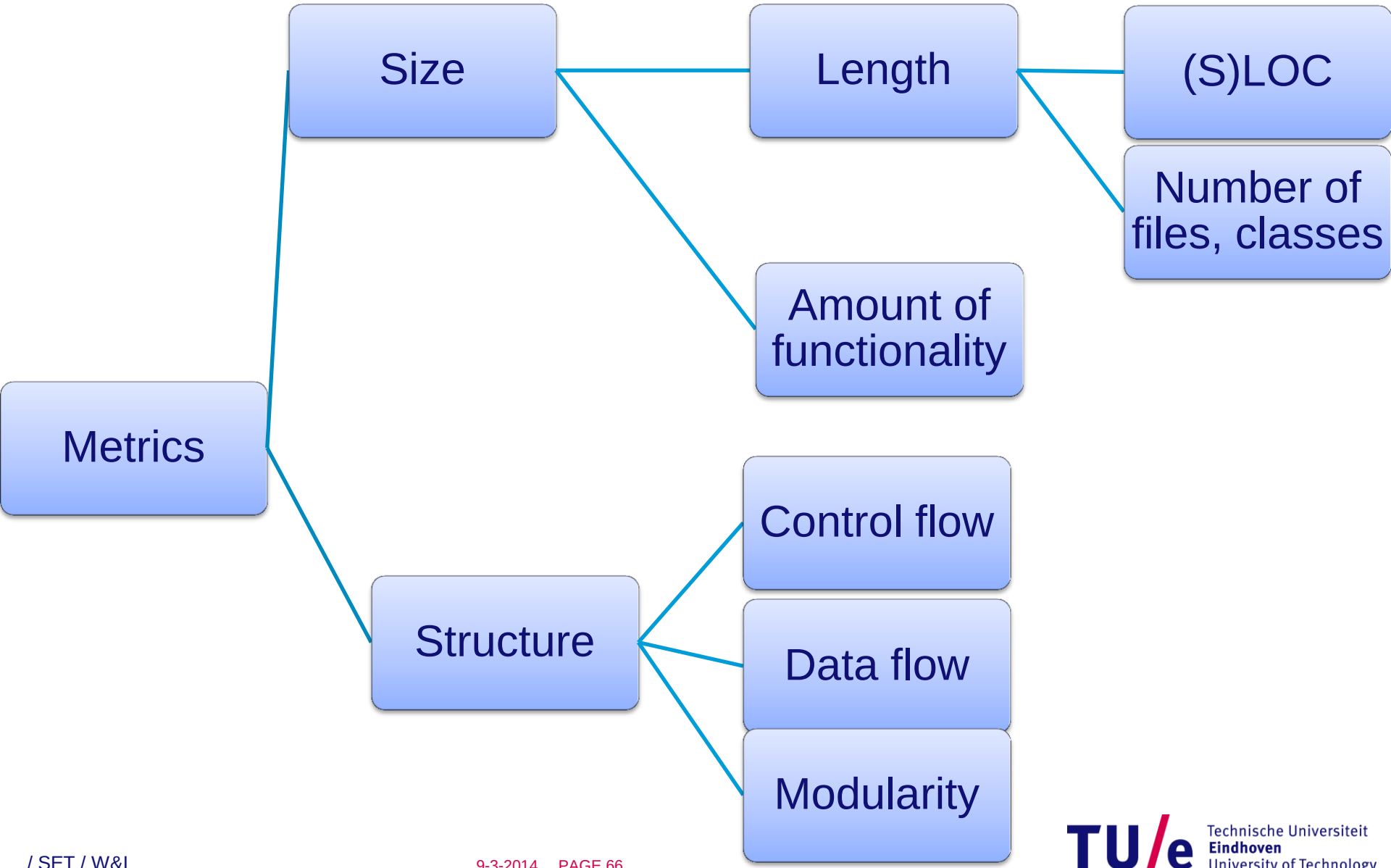
Metrics and scales

- **What metrics have we seen so far?**
  - **Size: LOC, SLOC**
  - **Code duplication: POP, RNF, …**
  - **Requirements: Flesch-Kincaid grade level**

To what scale does it belong?

Nominal	Ordinal	Interval	Ratio	Absolute
Imple-mentation language	Priorities (high > middle > low)	Tempe-rature (°C, °F)	m ↔ ft	#developers

Classification of metrics [à la Fenton, Pfleeger 1996]



Program length (LOC)

- **Variants:**
  - **Total**
  - **Non-blank**
  - **SLOC (source LOC): Ignore comments and blank lines**
  - **LLOC (logical LOC): Number of program statements**

```
for (i = 0;
        i < 100;
            i += 1) {
      printf("hello");
}

/* An important loop */
```

Total LOC: 7
Non-blank LOC: 6
SLOC: 5
LLOC: 2 (for and printf)

Advantages of (S)LOC

- Related to Lehman's law of "continuous growth" (Law 6)

- Easy to calculate
  - LLOC is more difficult to determine (parser needed)
  - What happens with nested statements? for(i=0;i<10;i++)?

- Correlation with the #bugs
  - Moderate (0.4-0.5) [Rosenberg 1997, Zhang 2009]
  - Larger modules usually have more bugs
    - "Ranking ability of LOC" [Fenton and Ohlsson 2000 , Zhang 2009]
  - There are better (but more complex) ways to predict #bugs
  - Can be used to predict the development effort!

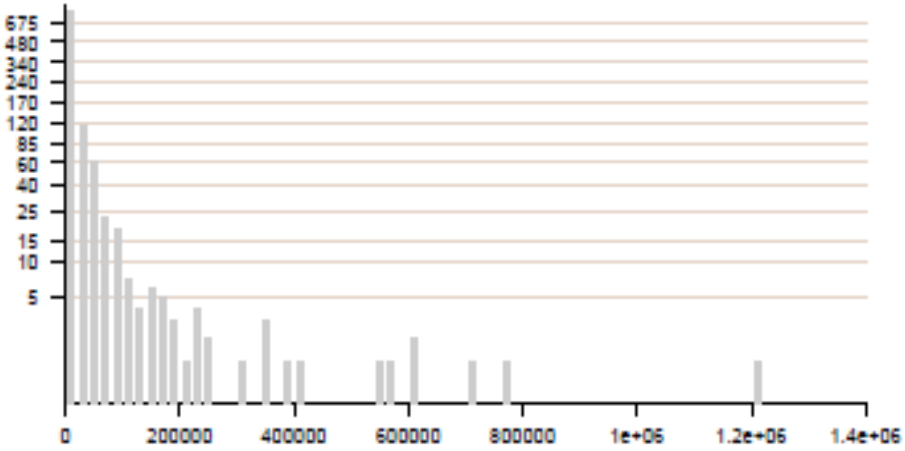
Disadvantages of (S)LOC

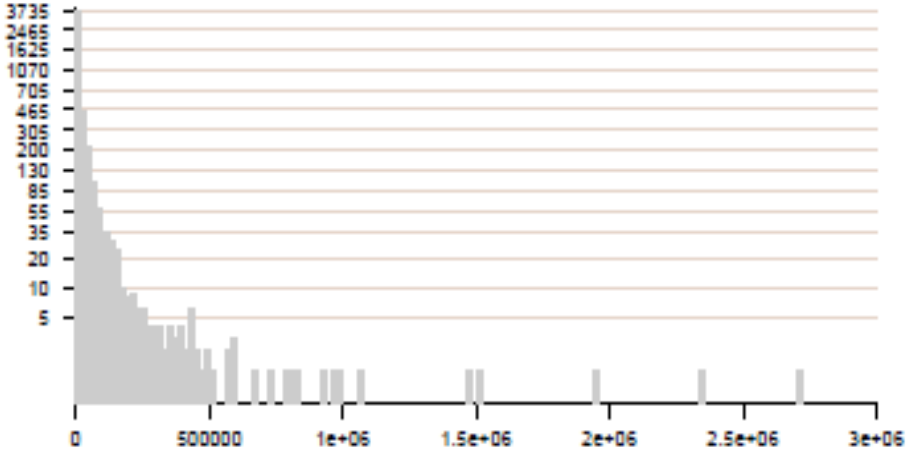
- **Ignores structure of the program**
  - **Program code is more than just text!**

- **Difficult to compare modules in different languages or written by different developers**
  - **Some languages are more verbose due to**
    - **Presence/absence of “built-in” functionality**
    - **Structural verbosity (e.g., .h in C)**
  - **Some developers are paid per LOC!**
  - **Hand-written vs. generated code**

(S)LOC distribution

Robles et al. 2006





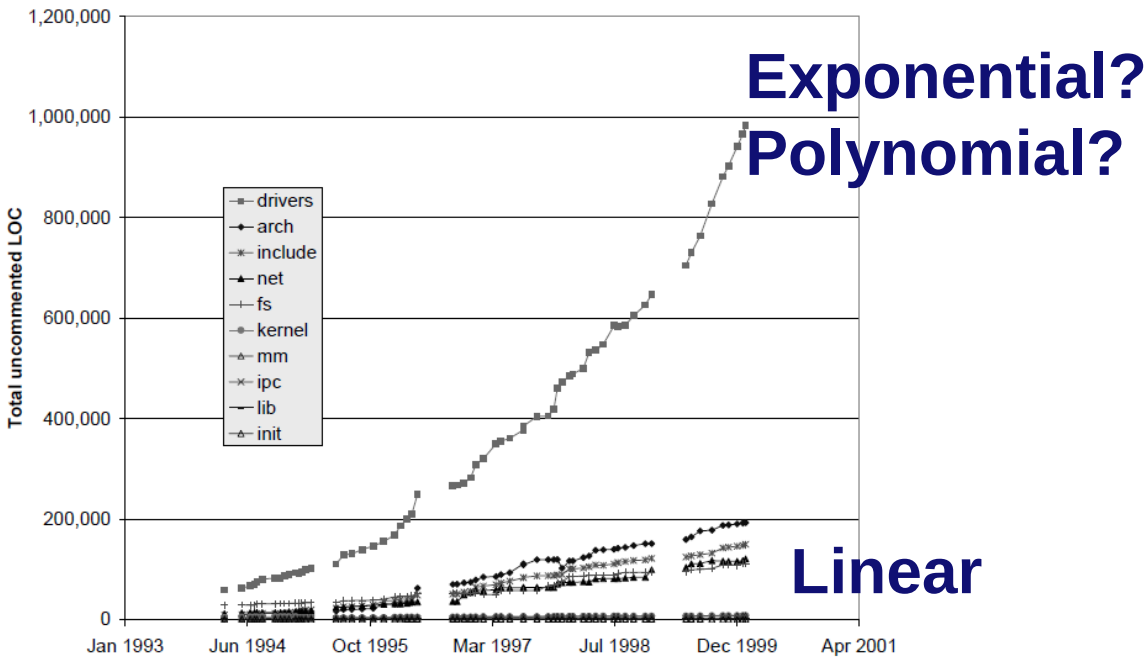
- **Distribution of SLOC in Debian 2.0 (left) and 3.0 (right)**
- **Controversy: log-normal or double Pareto?**
  - **Importance: knowing distribution one can estimate the probability to obtain files of a given size**
  - **Hence, to estimate size of the entire system**
  - **And the effort required (size ⇒ effort)**

What do we know about evolution of SLOC?

- **Related to Lehman's 6:**
  - **The functional capability <…> must be continually enhanced to maintain user satisfaction over system lifetime.**
  - **Earlier versions: "size".**

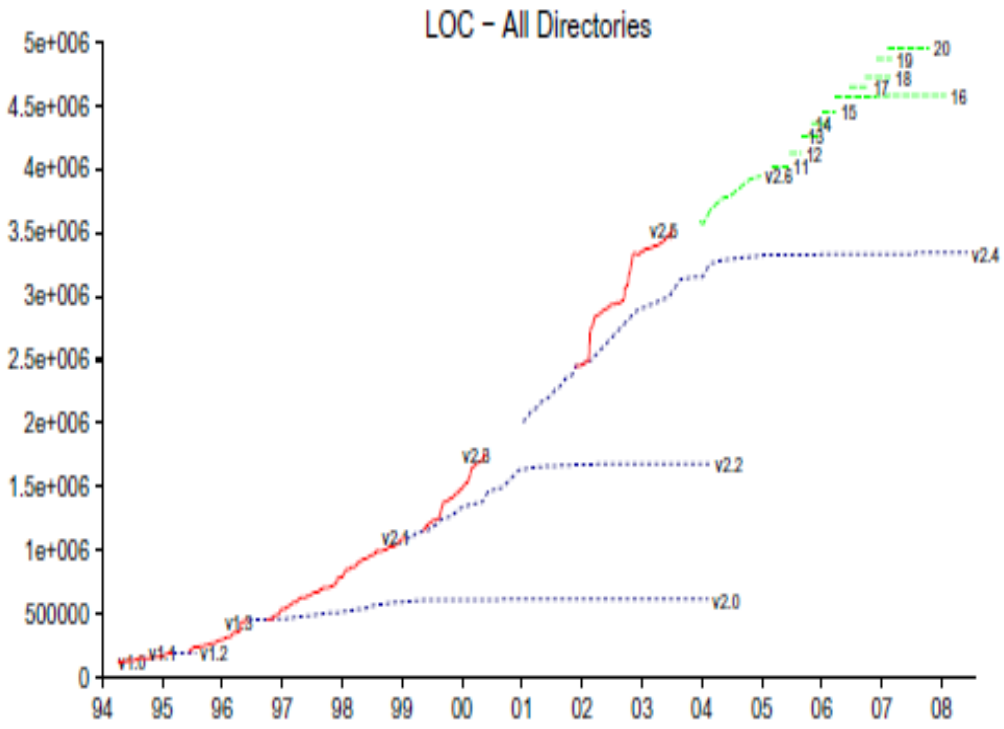
- **Also related to Lehman's 5:**
  - **In general, the incremental growth (growth rate trend) of E-type systems is constrained by the need to maintain familiarity.**
  - **Lehman interpreted this as linear growth**

What do we know about evolution of SLOC?



- Godfrey and Tu: superlinear growth is typical for OS
  - Koch 2007: Quadratic growth is better for larger OS projects (study of 8621 OS projects on SourceForge)

LOC in Linux kernel



Superlinear up to 2.5, linear for 2.6

Scacchi – mix of superlinear and sublinear

Israeli, Feitelson:

- **Linux kernel**
- **Multiple versions and variants**
  - **Production (blue dashed)**
  - **Development (red)**
  - **Current 2.6 (green)**

(S)LOC: Summary

- Different variants: LOC, SLOC, LLOC
- Advantages:
  - Easy to compute, moderately correlates with #bugs
  - Can be used to estimate the development effort (more details on May 15)
- Disadvantages
  - Different programming languages and developers
  - Hand-written vs. generated code
- Distribution “exponential-like”
- Evolution:
  - Linear
  - Linux (other OS?): Superlinear
  - Mix

Length: #components

- Number of files, classes, packages
- Intuitive: “number of volumes in an encyclopaedia”

- Variants:
  - All files, classes, packages
  - No empty/library/third-party files, classes, packages
  - No nested/inner classes
  - No or only some auxiliary files (makefiles, header files)

- Correlation with the #post-release defects [Nagappan, Ball, Zeller 2006]
  - significant for modules A, B, C (strength:0.5-0.7), insignificant for modules D, E
  - for each module correlation with some other metrics!