2IS55 Software Evolution

# What can we learn from version control systems?

Alexander Serebrenik

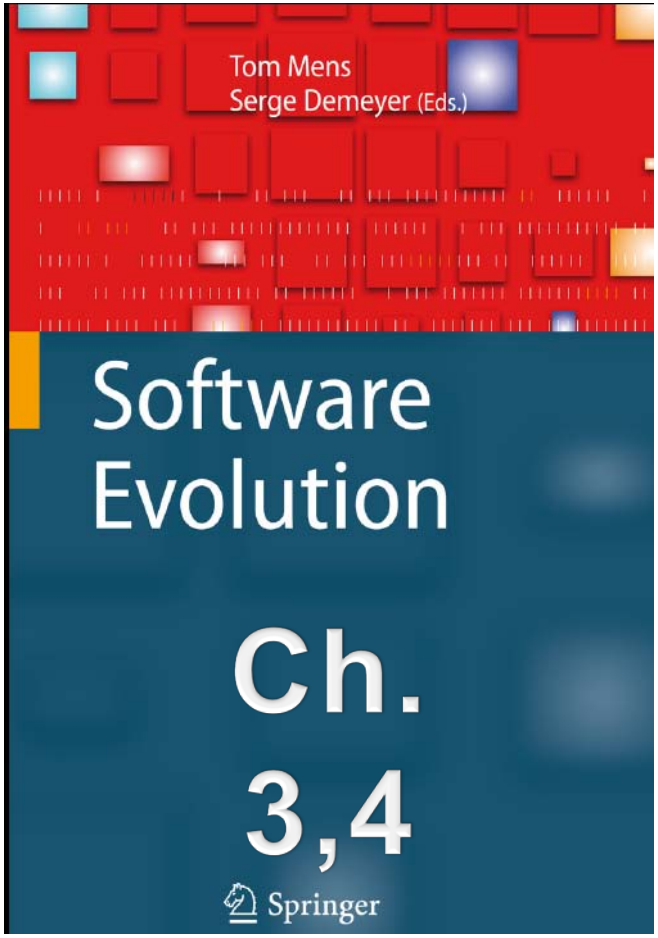
TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

# Assignments

- **Assignment 4:**
  - **Deadline: April 6**
  - **Questions?**
- **Assignment 5:**
  - **Published on Peach**
  - **Deadline: April 20**
- **Next quartile:**
  - **Starts from April 20**
  - **AUD 13 (May 4 included)**

# Sources


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

# Recap: Version control systems

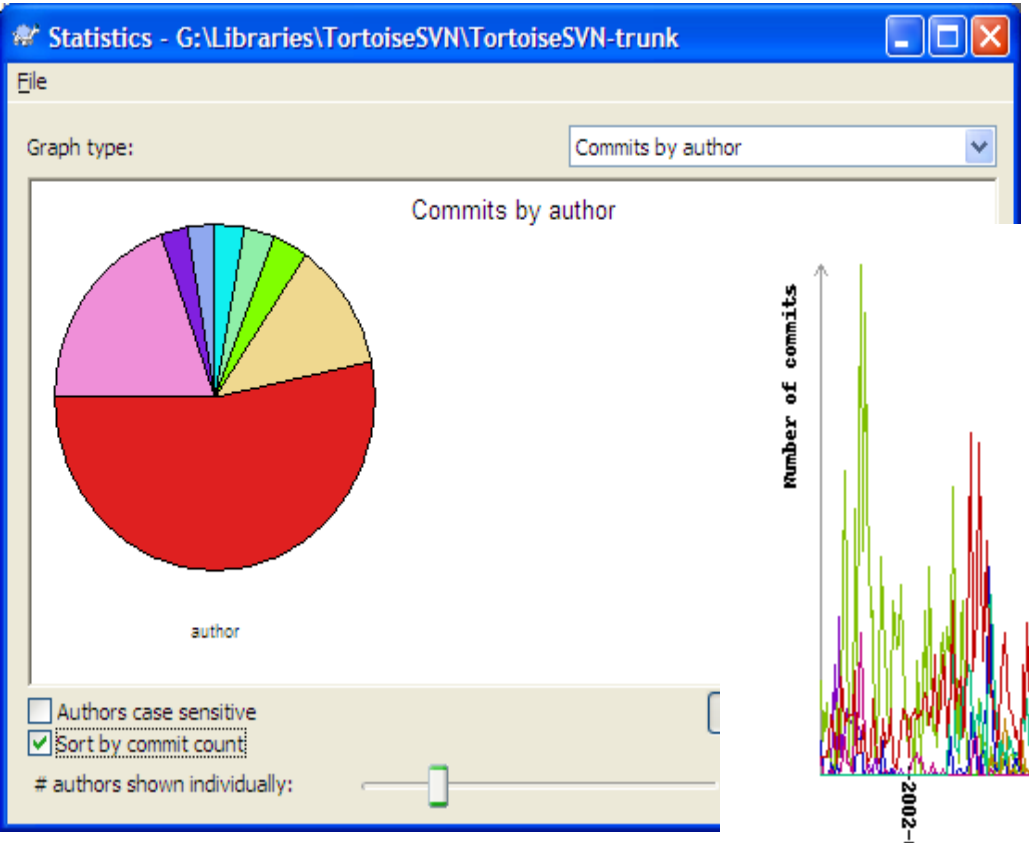
- Centralized vs. distributed
- File versioning (CVS) vs. product versioning

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

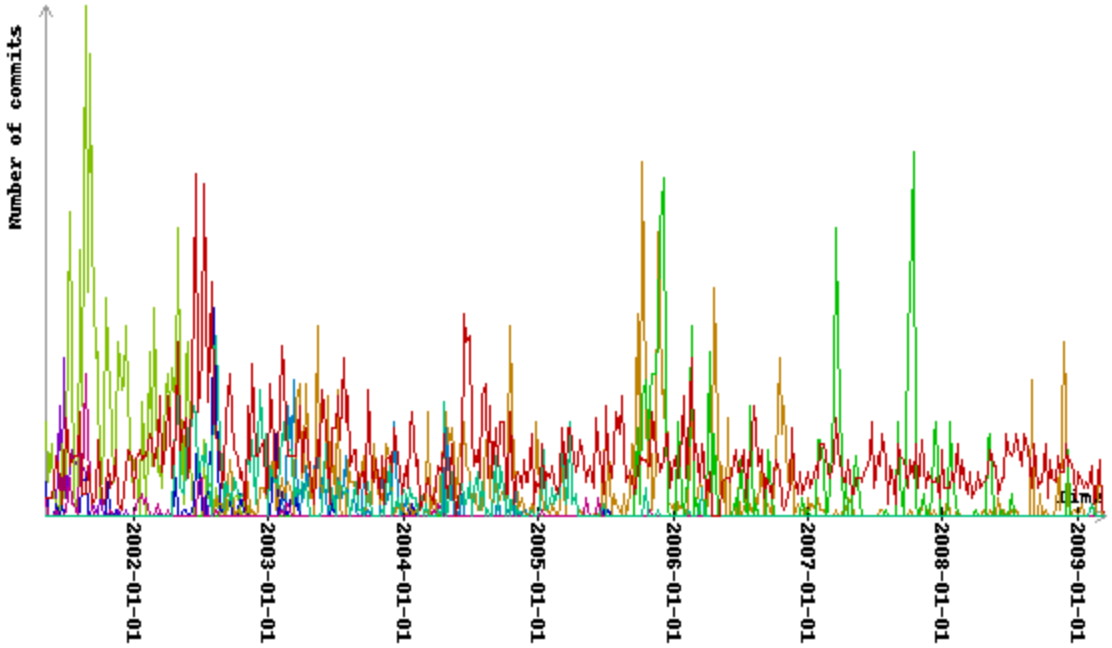
- What can we learn from this?
  - Humans
  - Files
  - Bugs

# What can we learn about the humans?

- Count commits per committer
- Look at how the counts evolve in time

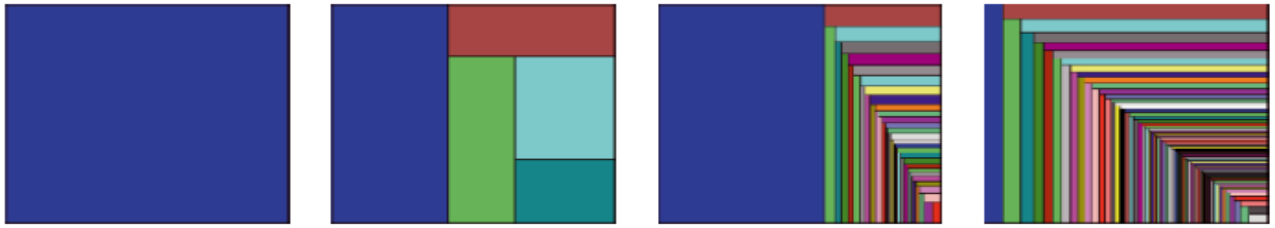



- One major committer?



# More refined way of counting: Per File

- **What developer worked on a file**
  - **Count pc(Alice): the % of commits on F made by Alice**
  - **Visualization (Fractal Figure)**
    - **pc is a relative area of a rectangle**



(a) One developer (b) Few balanced developers (c) One major developer (d) Many balanced developers

  - **Measure of "difference"**

$$1 - \sum_{c \in \text{committers}} pc^2(c)$$

  - **How does this measure behave for (a), (b), (c) and (d)?**

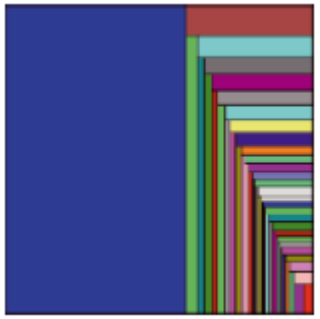


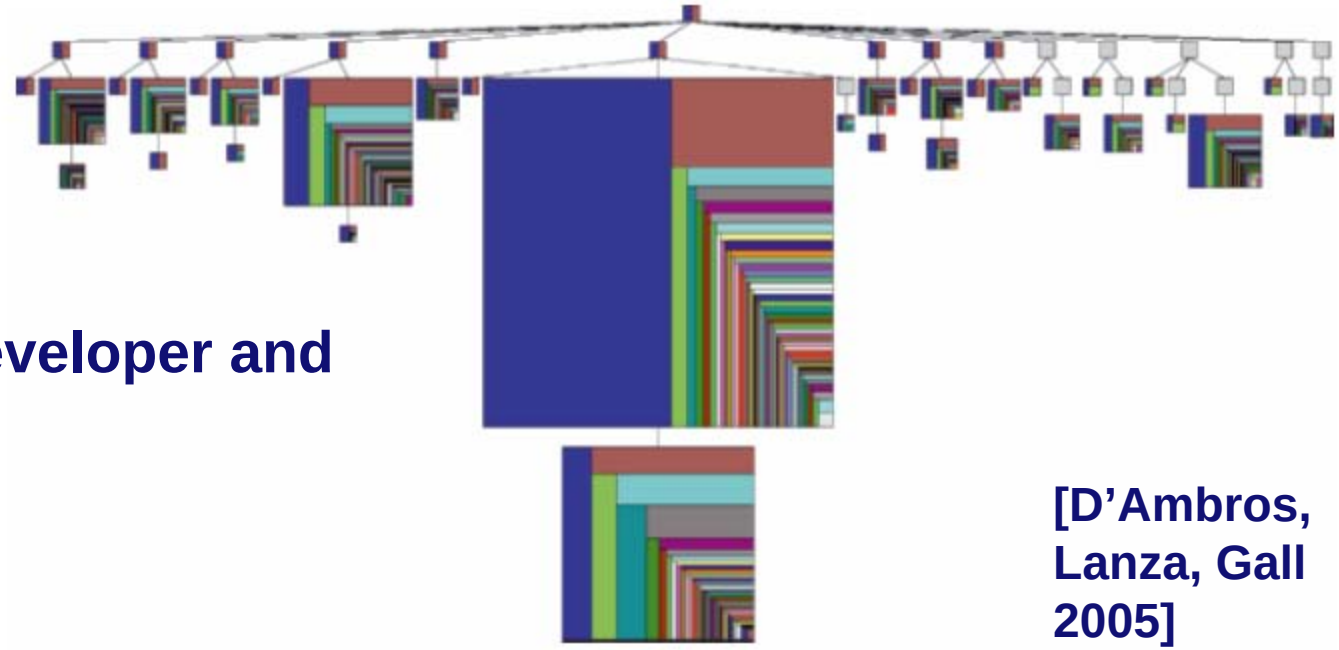
TU/e Technische Universiteit Eindhoven University of Technology

# Fractal Figures

- pc is a relative area
  - Blue vs. red, green, …
- Many options for absolute size
  - Number of changes
  - Size of an artefact (file, directory)
  - Number of bugs

One major developer and many bugs!





[D'Ambros, Lanza, Gall 2005]

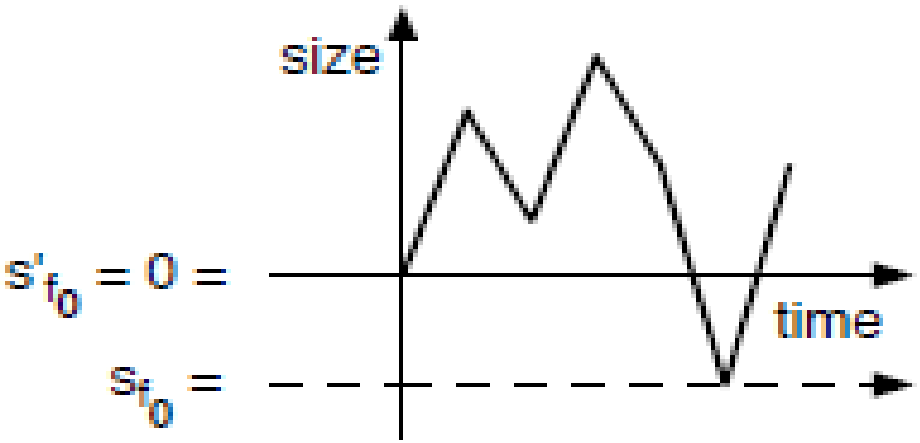
# … Size of an artefact?

- **Easy to determine if the code is available**
- **Can be estimated if only the log is available [Gîrba Kuhn Seeberger Ducasse 05]**

Working file: insert-msg.tcl
…
revision 1.2
date: 1999/03/05 07:23:11; author: philg; state: Exp; **lines: +30 -8**
changed the bboard to do generic file uploading (and fixed Ben's broken image uploading stuff)

**≥ 8 lines before**

**≥30 lines after**




$$s'_{f_0} := 0$$

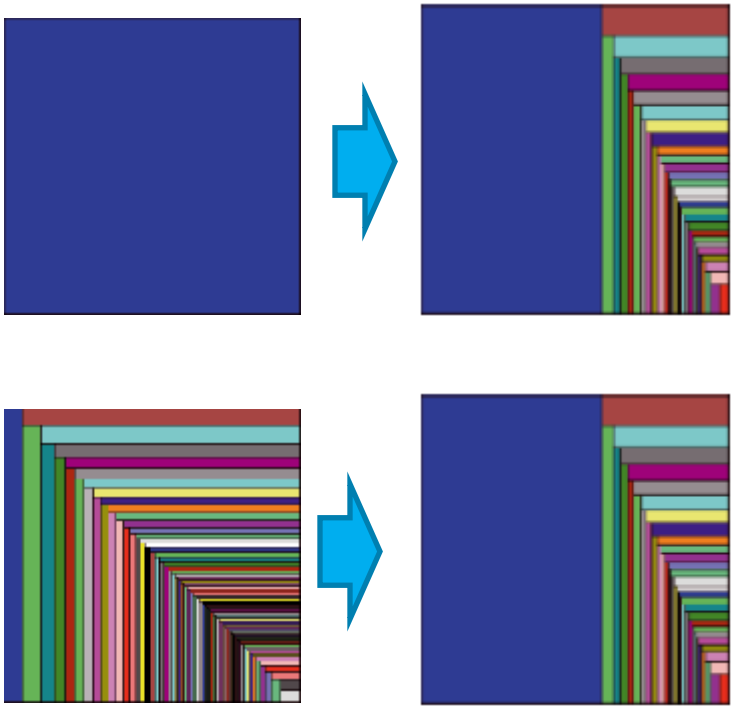
$$s'_{f_n} := s'_{f_{n-1}} + a_{f_{n-1}} - r_{f_n}$$

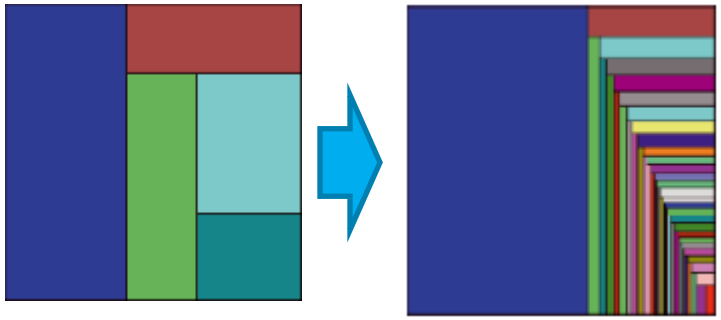
$$s_{f_0} := |min\{s'_x\}|$$

$$s_{f_n} := s_{f_{n-1}} + a_{f_n} - r_{f_n}$$

# However we still have only a static view…

- How does the picture evolve in time?

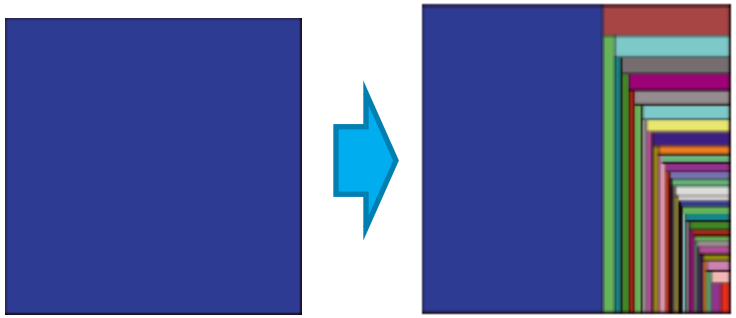


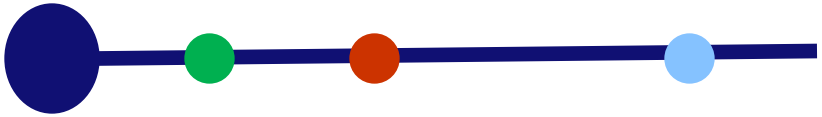


- Solutions:
  - Graph of fractal values
  - Ownership maps

# Ownership maps [Gîrba Kuhn Seeberger Ducasse 05]

- **Owner of…**
  - **line = last committer of this line**
  - **file = owns the major part of the lines**
    - **requires calculation of the file size**
    - **can be estimated from the log**

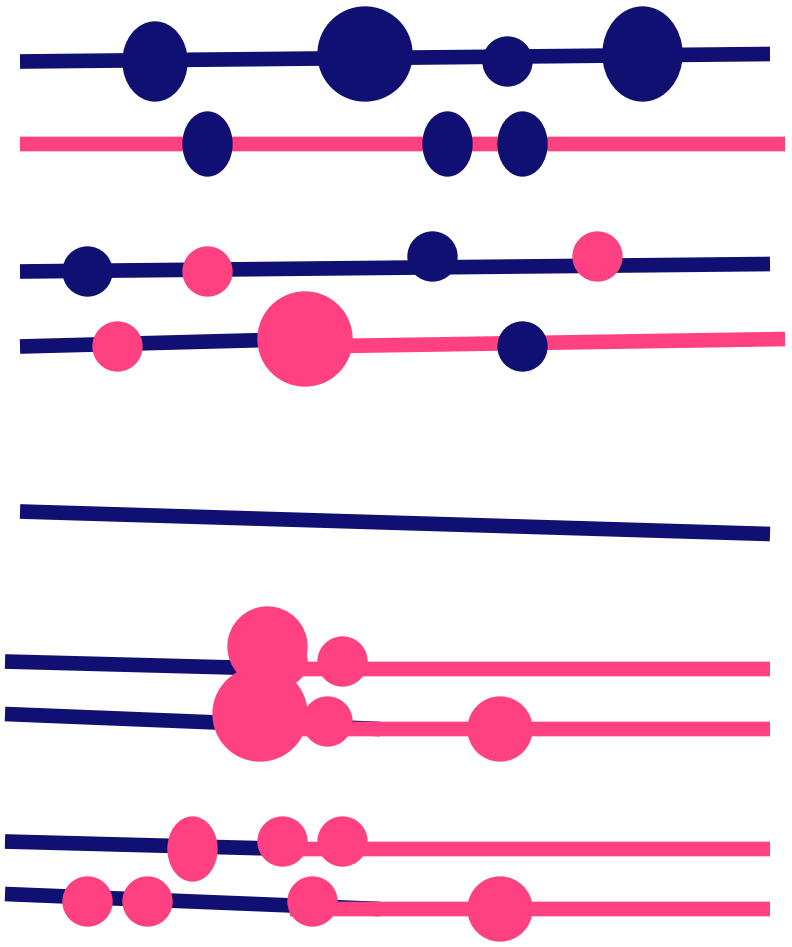




- **Colour = committer**
- **Circle = commit**
- **Line = owner**
- **Timeline**
- **Size = proportion of change**

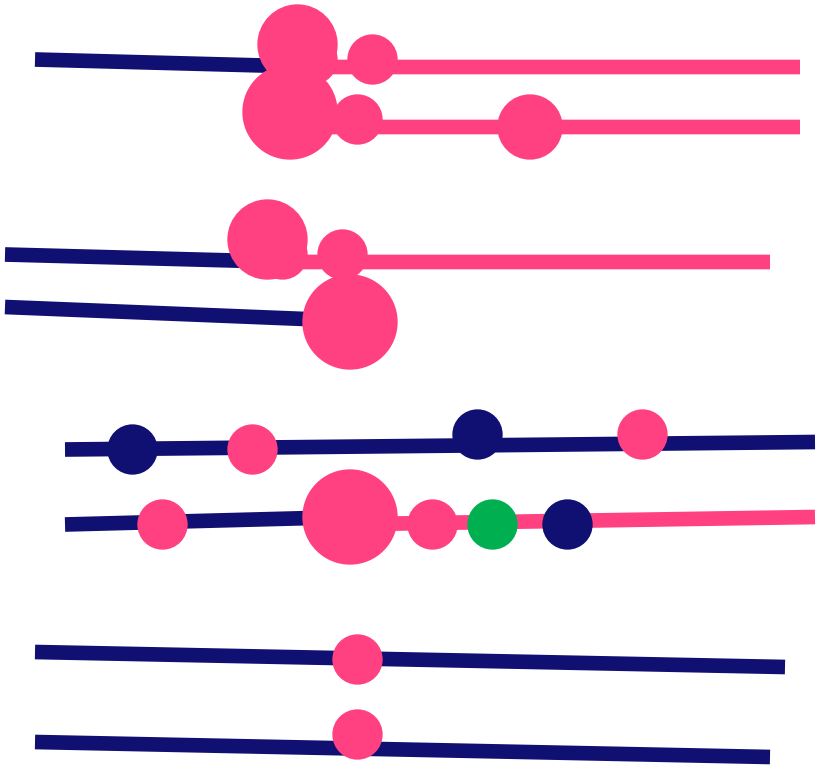
# Development patterns

- **Monologue**
- **Dialogue**
  - **Teamwork (quick succession)**
- **Silence**
- **Takeover**
  - **Epilogue (Takeover + Silence)**
- **Familiarization**



# Development patterns (continued)

- Expansion
- Cleaning
- Bug fix
- Edit
  - Epilogue (Edit + Silence)

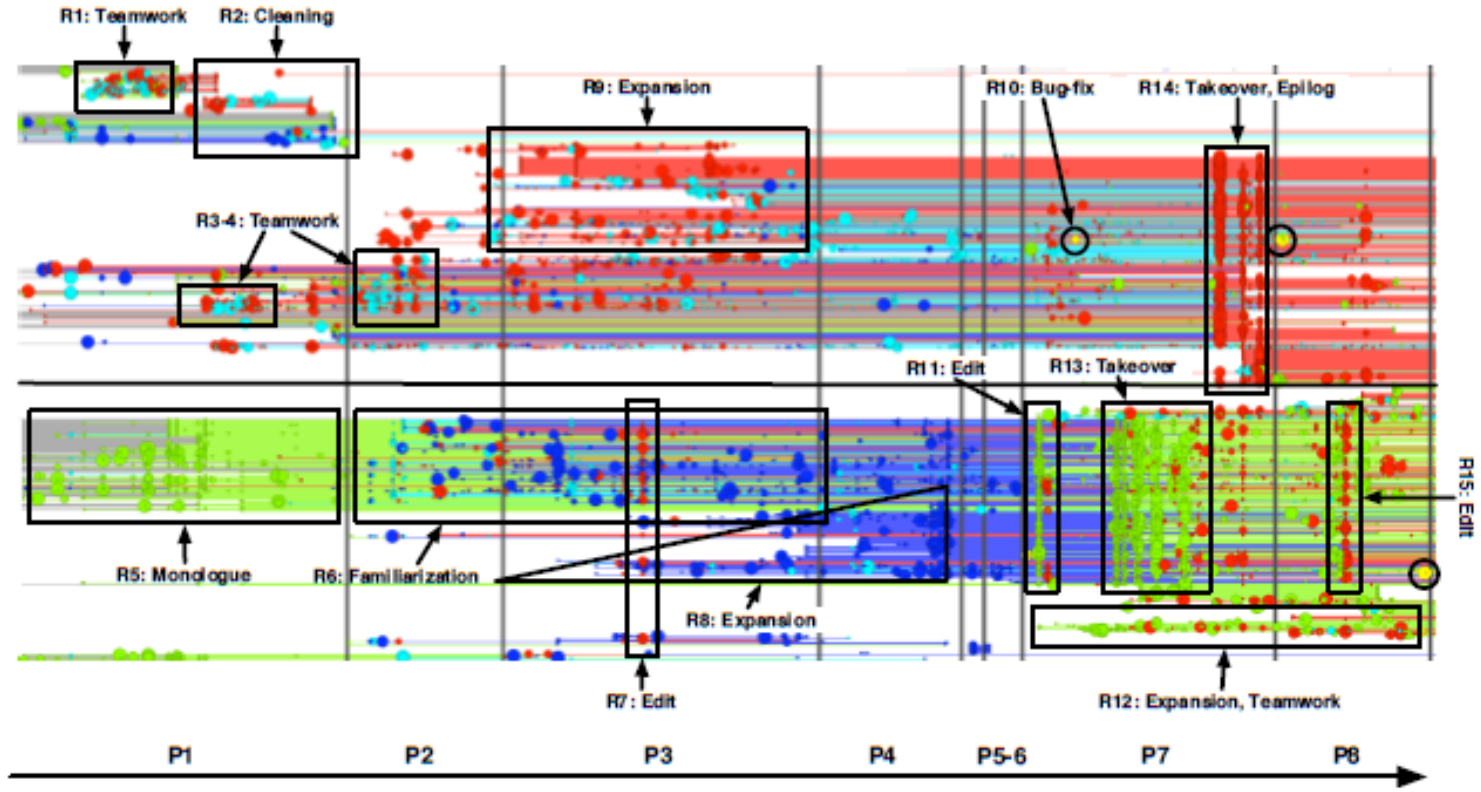


# Experiment: Outsight

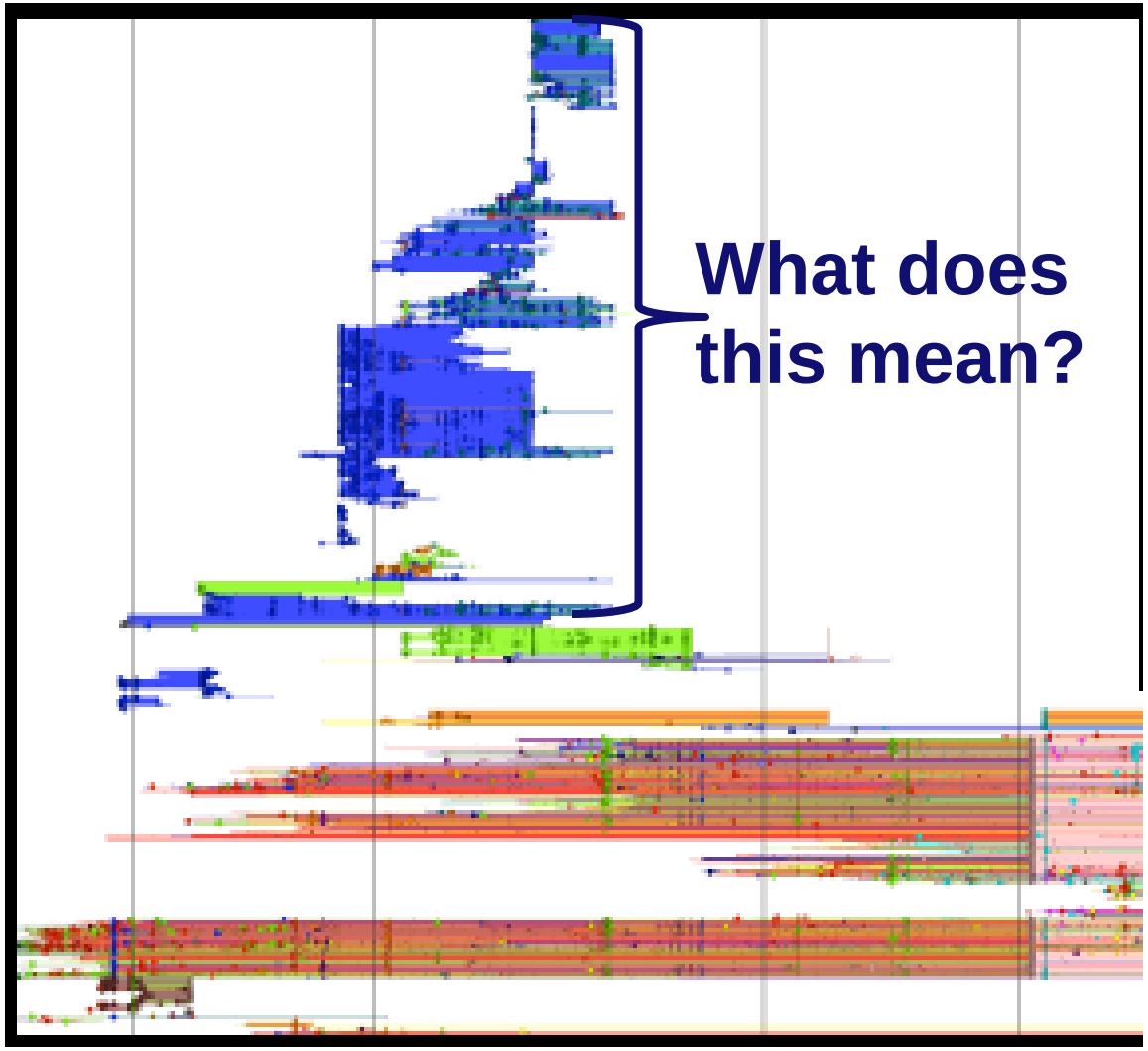
- Commercial application, 500 Java classes, 500 JSP
- 8 three-months periods

- How many developers are there?

- If you had questions about the system, whom would you ask?



# Ant




**Subproject (Myrmidon) that was intended as a successor for Ant.**

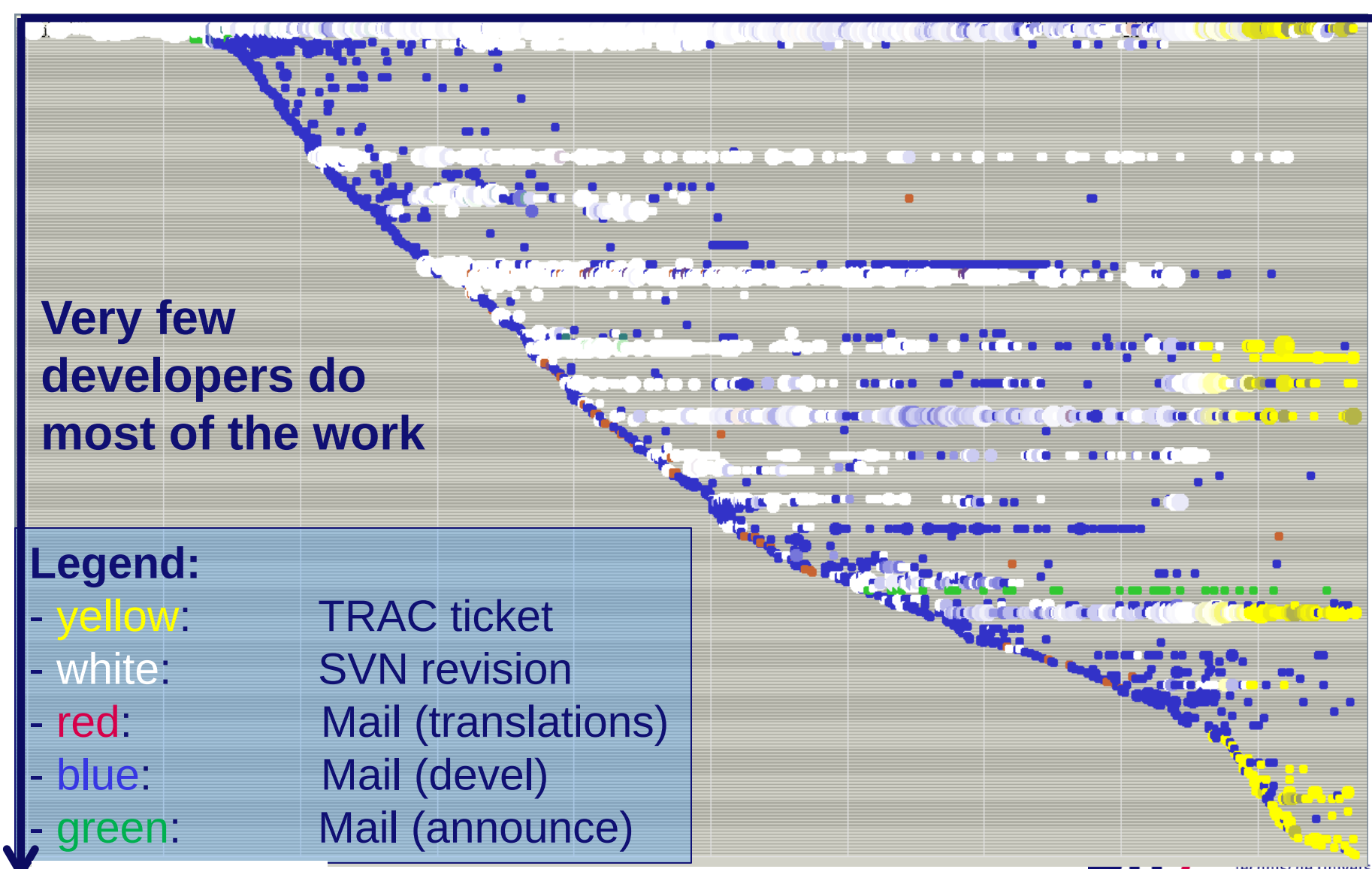
**Pattern common to Open Source**

**Subprojects**

- **Cease**
- **Split**
- **Integrate in the main line**

# How do people work? [Poncin et al. 2011]

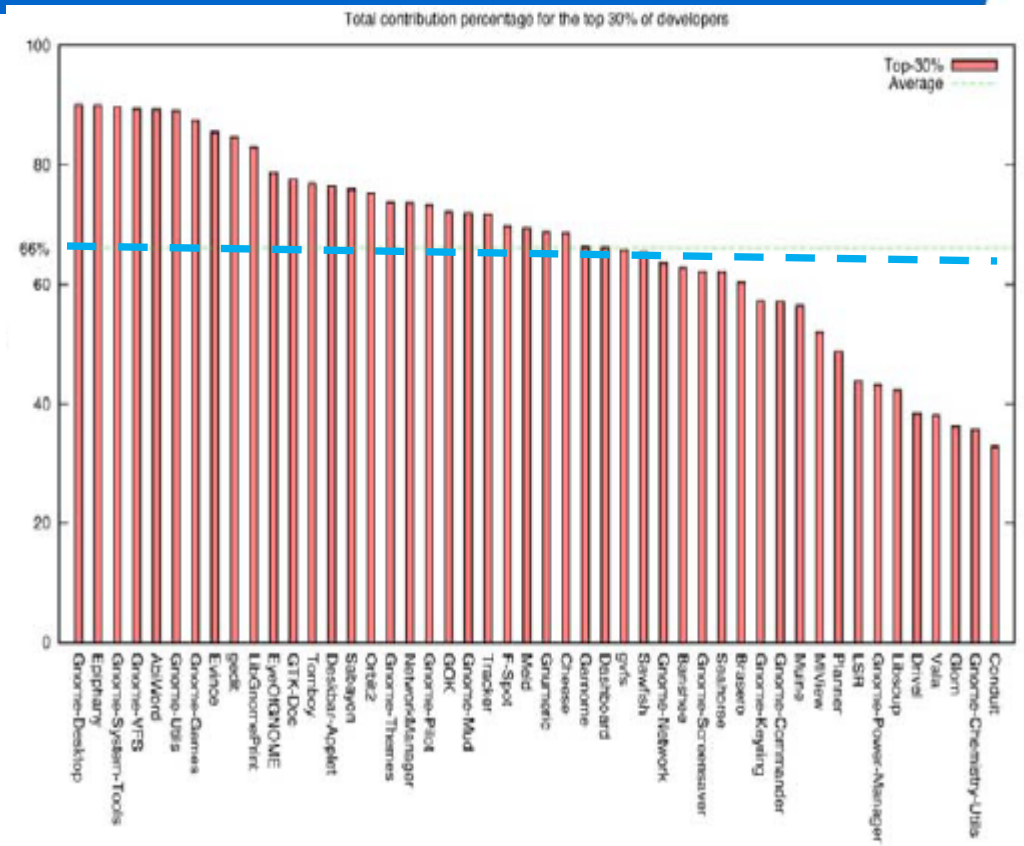
Time




Developers

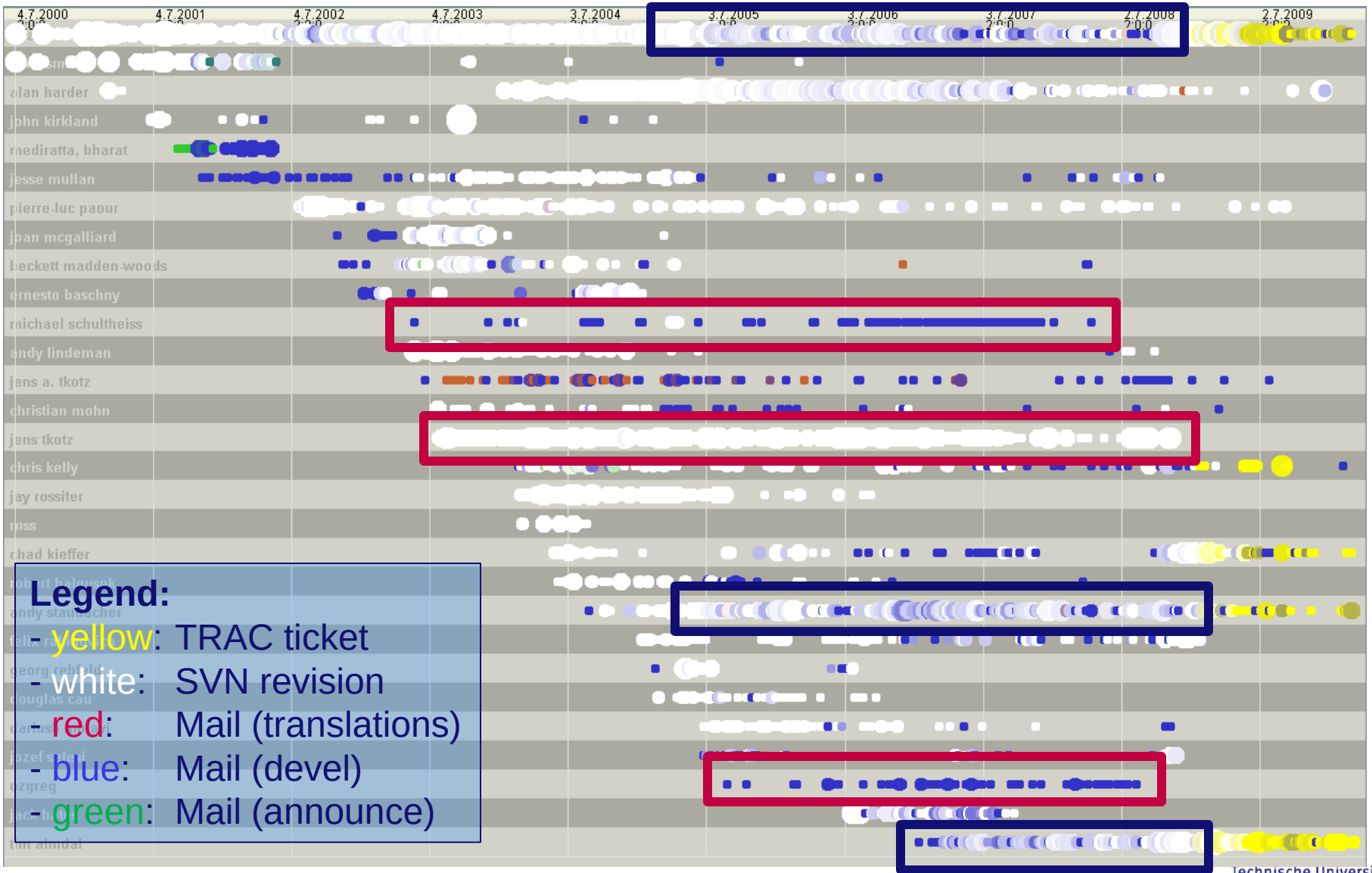
# “Very few developers do most of the work”

- “Pareto principle” 20/80
- Quite common for software metrics
  - More precise descriptions of the distribution are possible
  - Even for LOC no agreement on the precise distribution



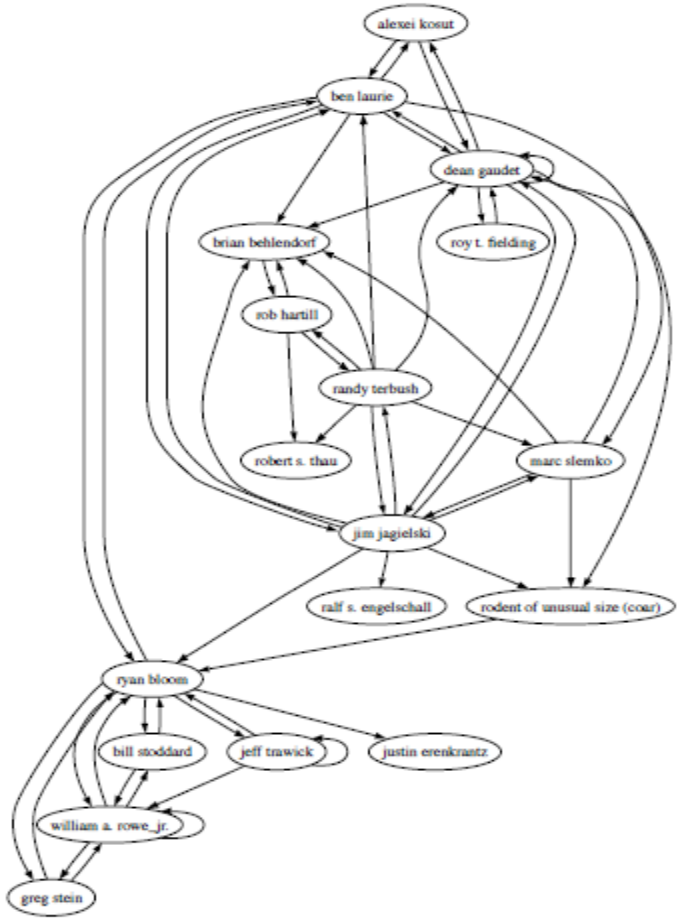

**Contribution of 30% most prolific developers in different GNOME projects [Kalliamvakou, Gousios, Spinellis, Pouloudi, 2009]**

# FRASR: Who does what?



# All developers are equal, but some are more equal than others [Bird et al. 2006]

- **Mail archive vs. version control**
  - **Without commit rights: “non-developers”**
  - **With commit rights: some commit more often**




**Mail communication (arrow = at least 150 mails send)**

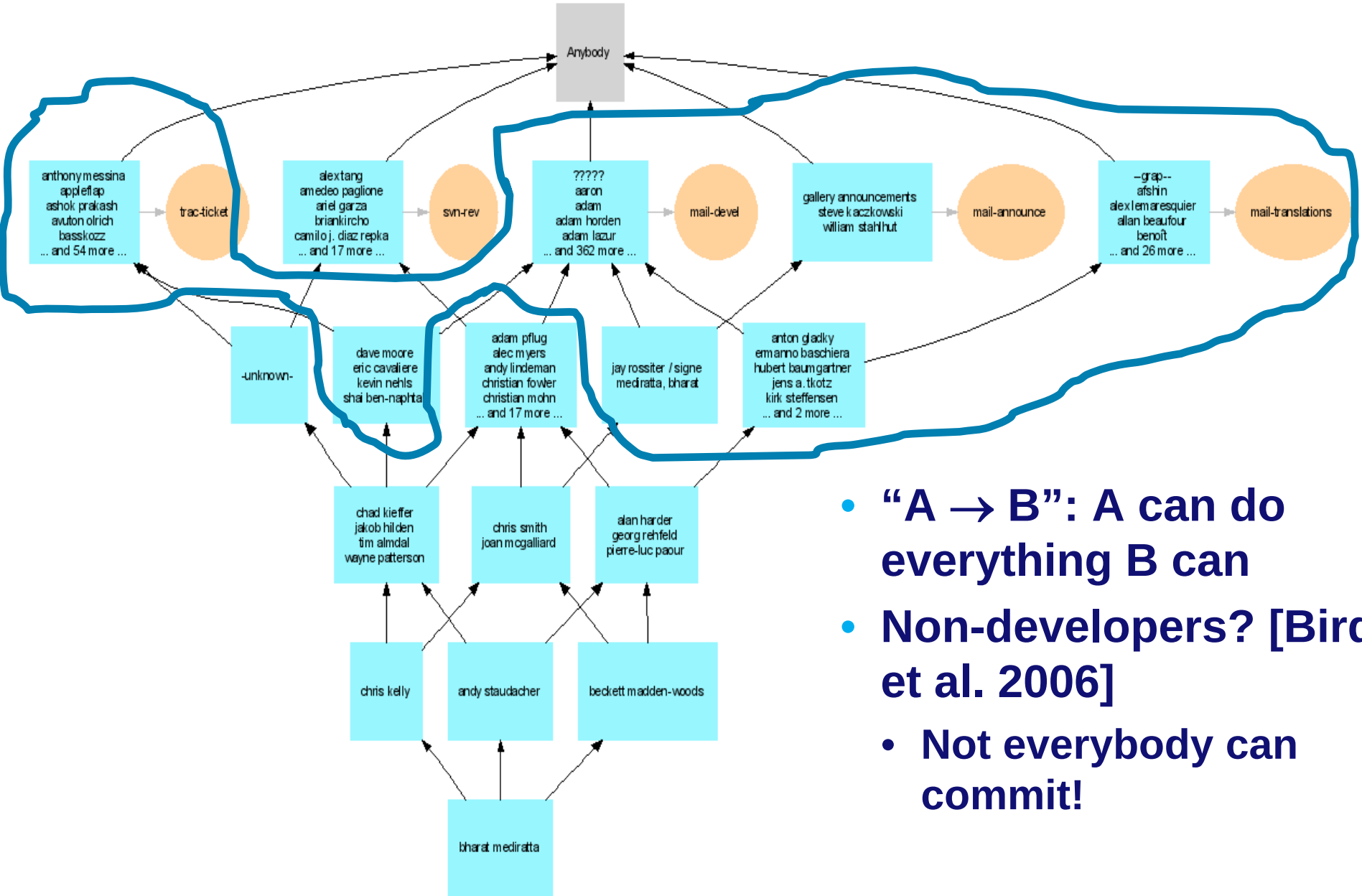
**Conclusion 1: Developers are more active than non-developers**

**Conclusion 2: Correlation between the number of commits and the “centrality” of the developer**



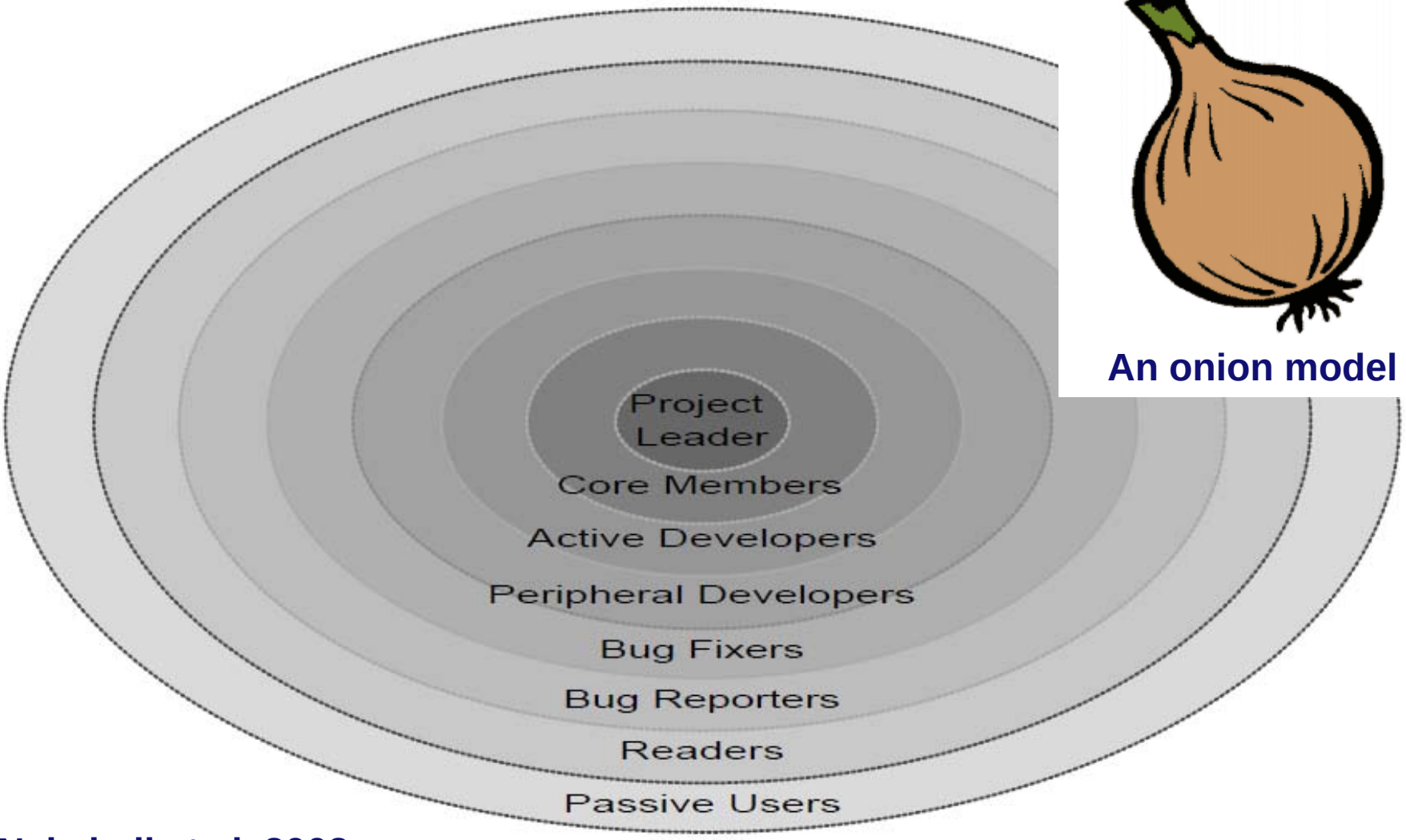
TU/e Technische Universiteit Eindhoven University of Technology

# More refined developers classification is possible! [Wouter Poncin]




- **"A → B": A can do everything B can**
- **Non-developers? [Bird et al. 2006]**
  - **Not everybody can commit!**

# What kind of roles do the developers play?

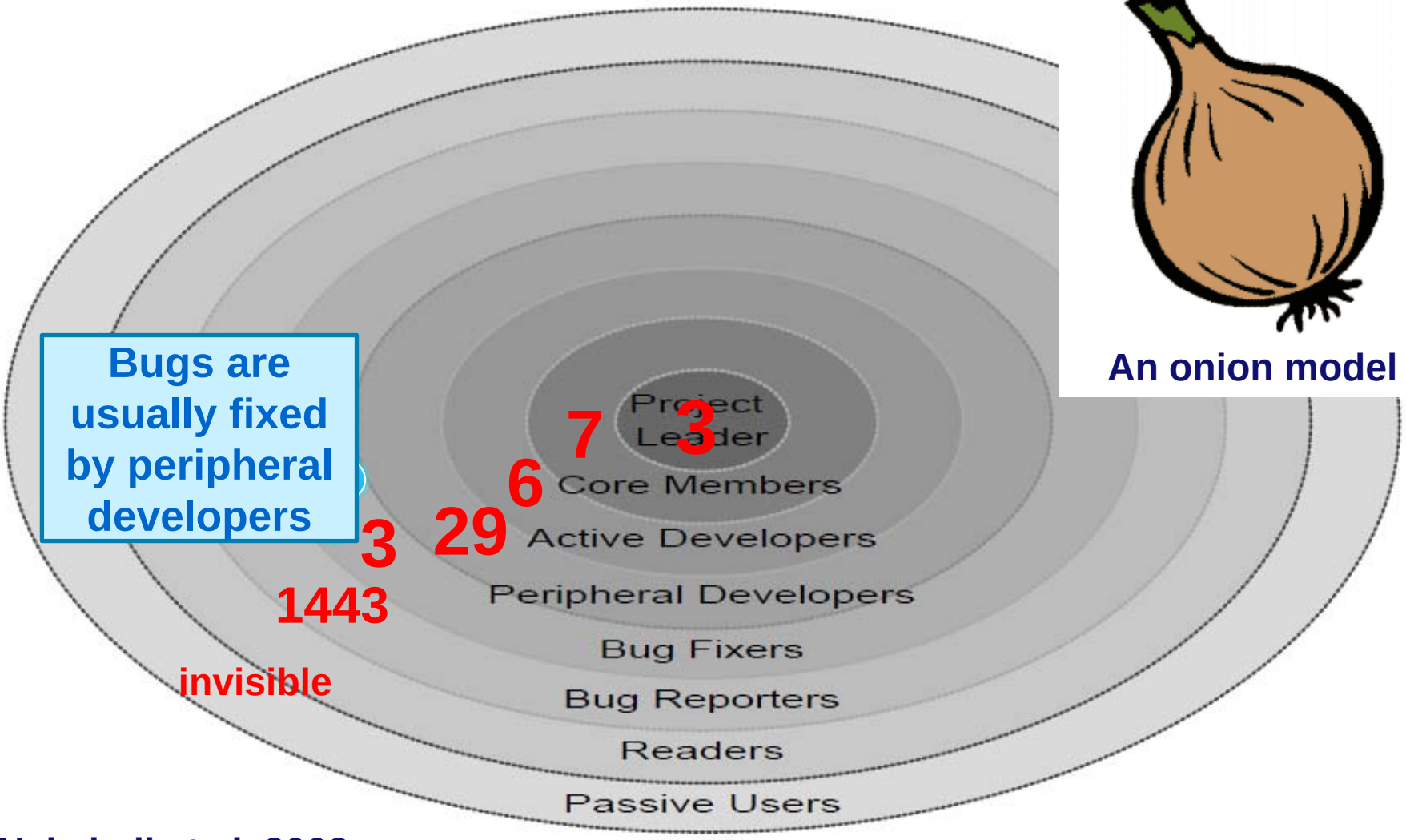





**An onion model**

**Nakakoji et al. 2002**

# Onion in aMSN



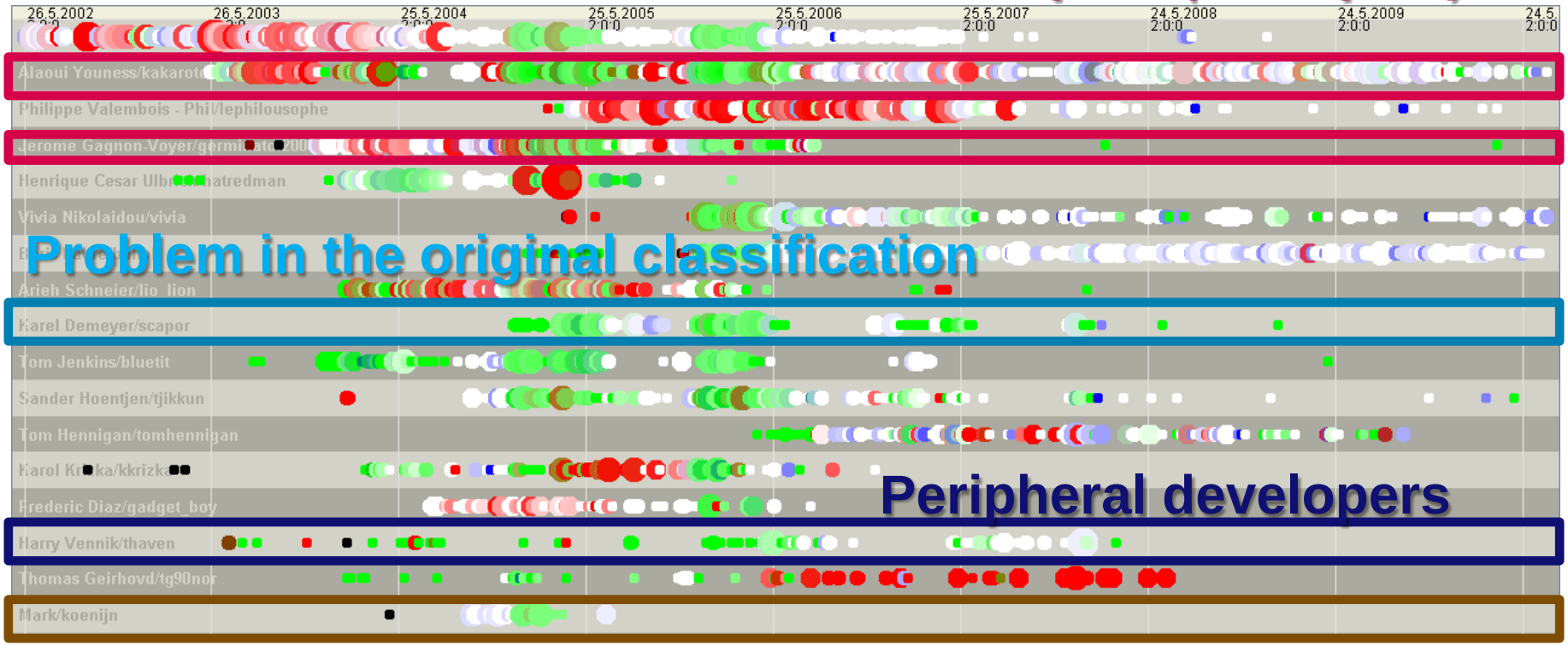

**Bugs are usually fixed by peripheral developers**

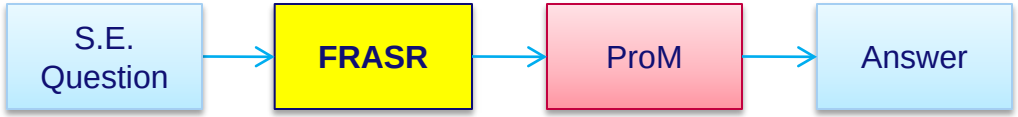


**An onion model**

**Nakakoji et al. 2002**

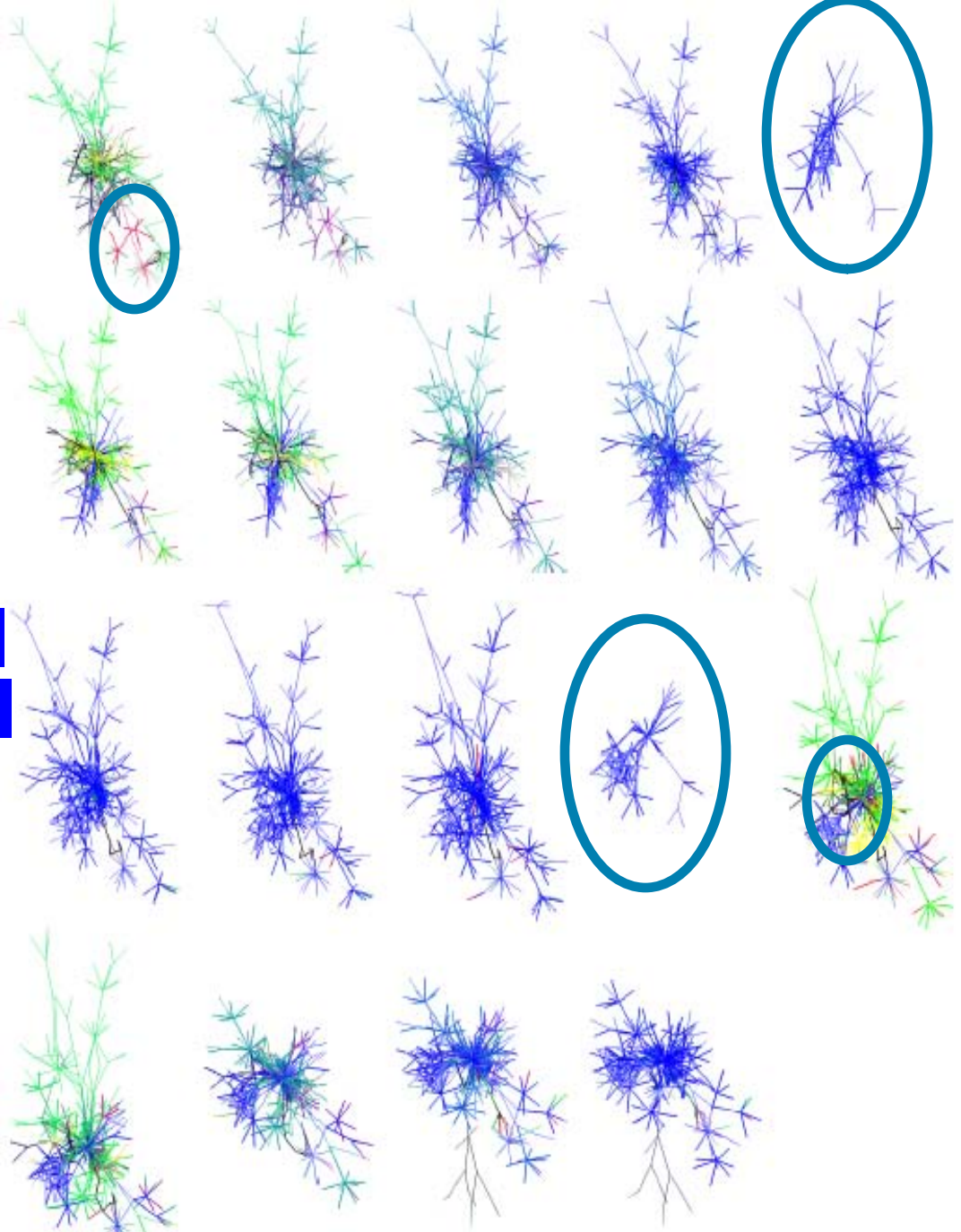
# Nakakoji et al. as a case of



# Developer activity is not constant

- Inheritance graph
- Colours correspond to developers and “cool down” if not updated
  - Dev 1
  - Dev 2
  - Other developers



**Broken commits**

**Dev 2 is more central: architect, Dev 1: programmer**

Collberg,Kobourov,Nagra,Pitts,Wampler 2003

# Users in mail archives, version control systems, etc.

- Multiple aliases
  - a.serebrenik@tue.nl
  - aserebre@win.tue.nl
  - aserebrenik@yahoo.com
  - alex@alum.cs.huji.ac.il
- Can be worse:
  - Ken Coar a.k.a. “Rodent of unusual size”
- Alias resolution problem
  - “From : Serebrenik, A. <a.serebrenik_at_tue.nl>”
  - “From: aserebre at win.tue.nl (A Serebrenik)”
  - But sometimes <a.serebrenik@xxxxxx>

# Clustering names and e-mails

- **Normalize names:**
  - **Remove punctuation and suffixes (“jr.”), reduce spaces and drop generic terms (“admin”, “support”)**
  - **Separate first name and last name**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| S | a | t | u | r | d | a | y |
| S | a | t | u | n | d | a | y |
| | S | a | u | n | d | a | y |
| | | S | u | n | d | a | y |

## 3 similarity measures

- **Similarity of names**
  - **Levenshtein distance**
  - **Number of characters added, removed or modified**
  - **Names are similar if**
    - **either the full names are similar**
    - **or both the first and last names are similar**

# Clustering names and e-mails

- Similarity of names and mails
  - The prefix (before @)
    - Contains the first and the last names
    - Robles: Contains the first or the last name and the first letter of the other one
- Similarity of mails
  - Levenshtein distance on prefixes
- Cumulative similarity – maximal of the three

- Clustering based on the cumulative similarity
  - Large clusters
  - Human inspection and post-processing
    - It is easier for humans to split large clusters than to combine small ones

**Still an heuristics!**

# How to calculate the Levenshtein distance?

- Words X (n characters), Y (m characters)
- Data structure C[0..n,0..m]
- Init: C[i,0]=i, C[0,j]=j for any i and j

| C | | S | a | t | u | r | d | a | y |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| S | 1 | | | | | | | | |
| u | 2 | | | | | | | | |
| n | 3 | | | | | | | | |
| d | 4 | | | | | | | | |
| a | 5 | | | | | | | | |
| y | 6 | | | | | | | | |

Similar to the longest common sequence (diff)

# How to calculate the Levenshtein distance?

- **For every i and every j**
  - **If X[i]=Y[j] then C[i,j]=C[i-1,j-1]**
  - **Else C[i,j]=min(C[i-1,j]+1, // deletion**
    **C[i,j-1]+1, // insertion**
    **C[i-1,j-1]+1) // modification**

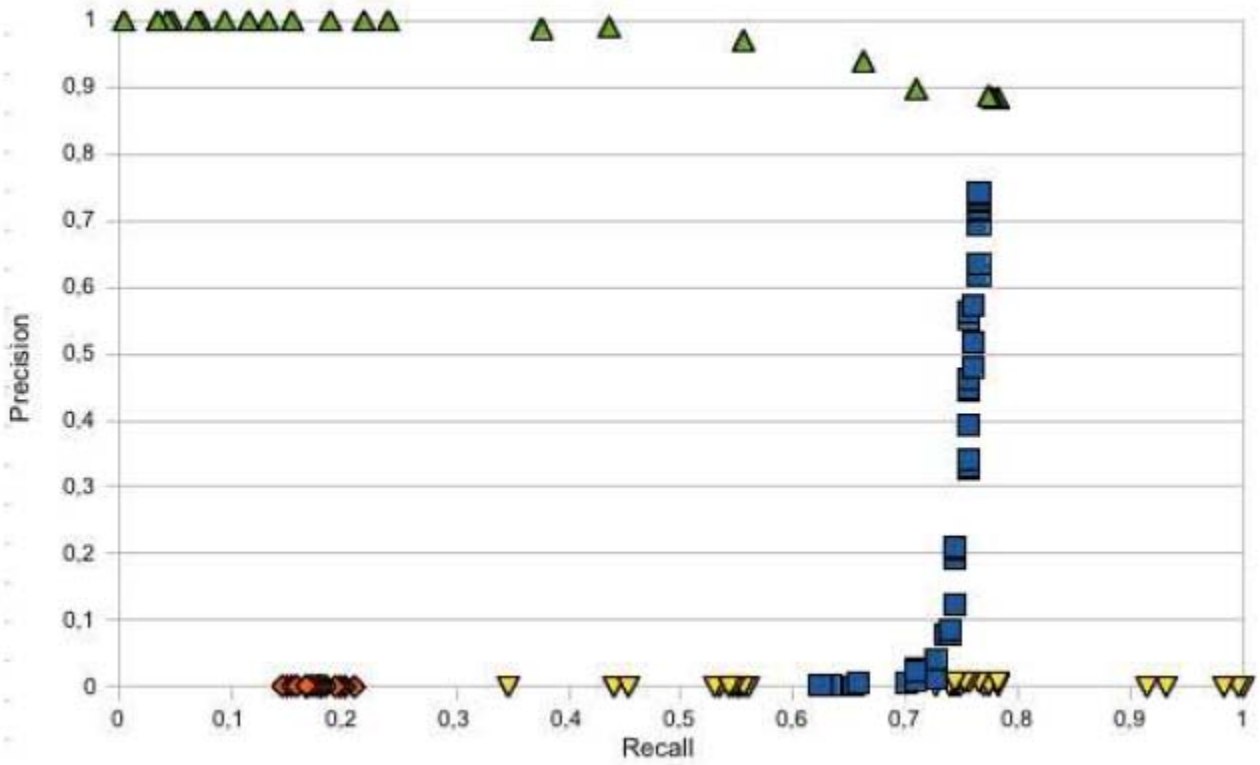
| C | | S | a | t | u | r | d | a | y |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| S | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| u | 2 | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 |
| n | 3 | 2 | 2 | 2 | 3 | 3 | 4 | 5 | 6 |
| d | 4 | 3 | 3 | 3 | 3 | 4 | 3 | 4 | 5 |
| a | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 | 4 |
| y | 6 | 5 | 4 | 4 | 5 | 5 | 5 | 4 | 3 |

**The Levenshtein distance!**



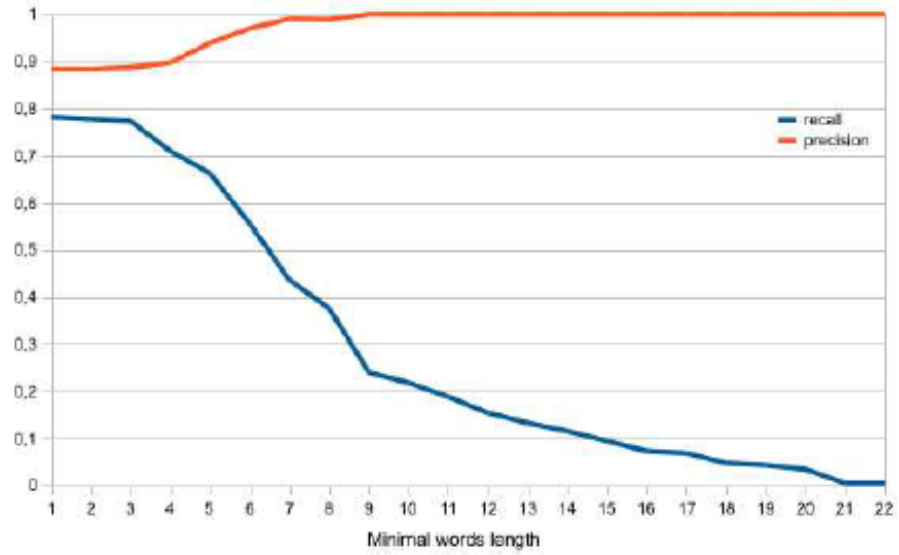
TU/e Technische Universiteit Eindhoven University of Technology

# How does it work in practice?

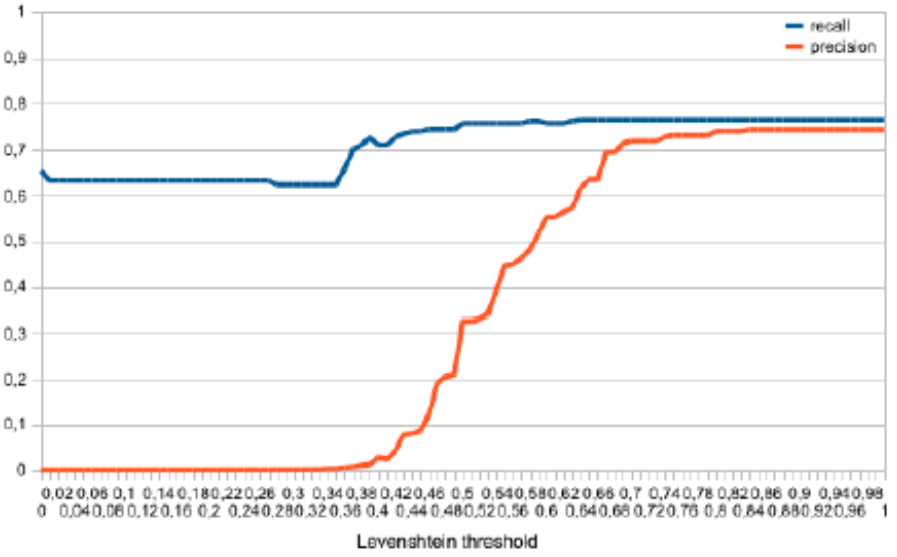



**Different points – different thresholds ("similar enough")**

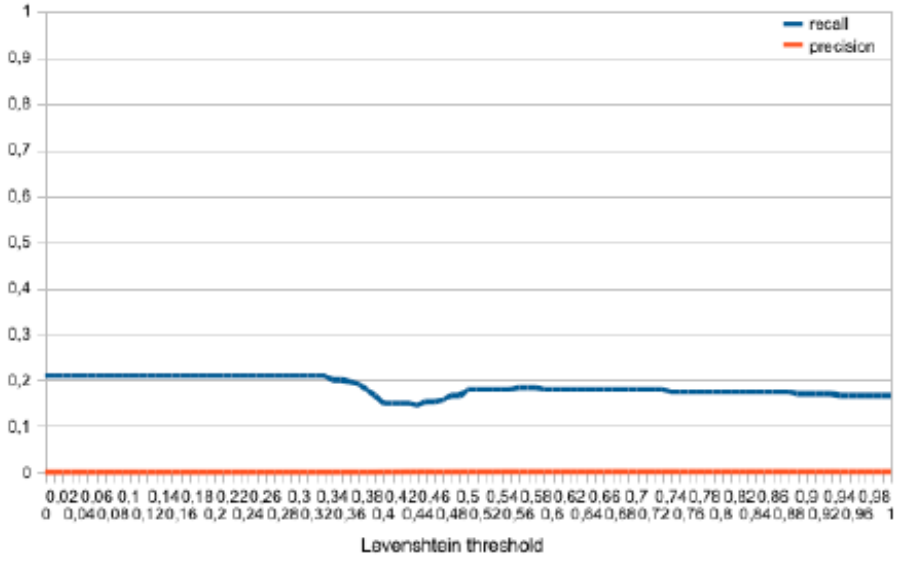
- **Simple: identical names, e-mail prefixes or user names**
- **Bird: discussed so far (no "jsmith")**
- **Bird extended: Bird + bug tracker names**
- **Improved: "jsmith", multiple parts, special characters**



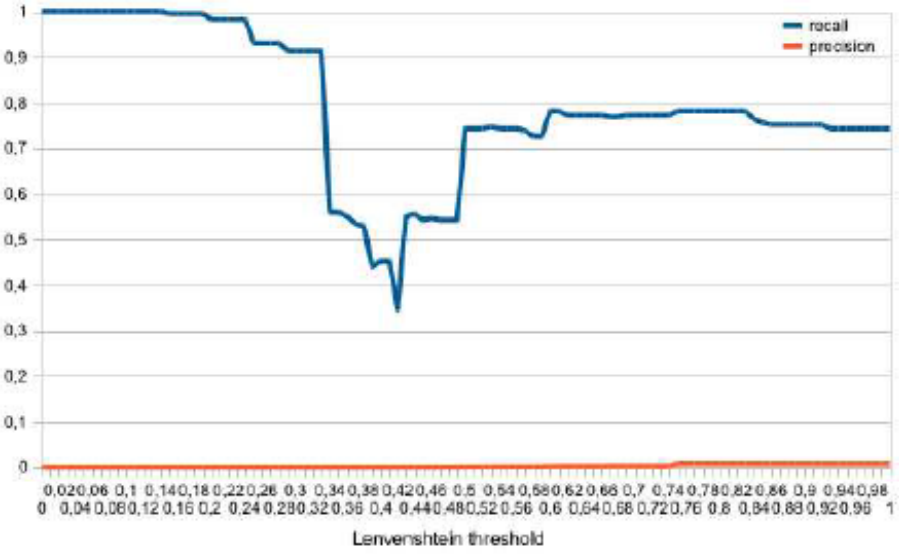

(a) Simple




(b) Improved



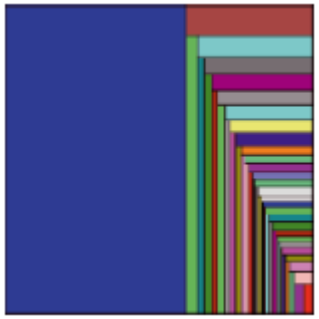

(c) Bird

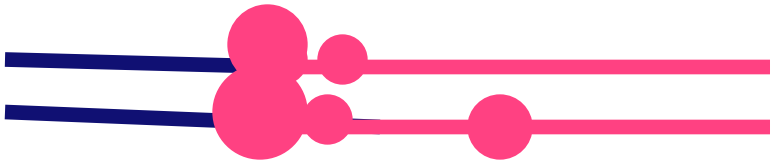



(d) Bird extended

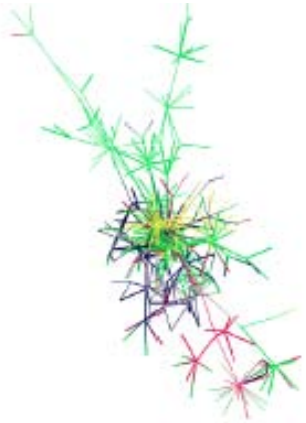
# What can we learn about humans?

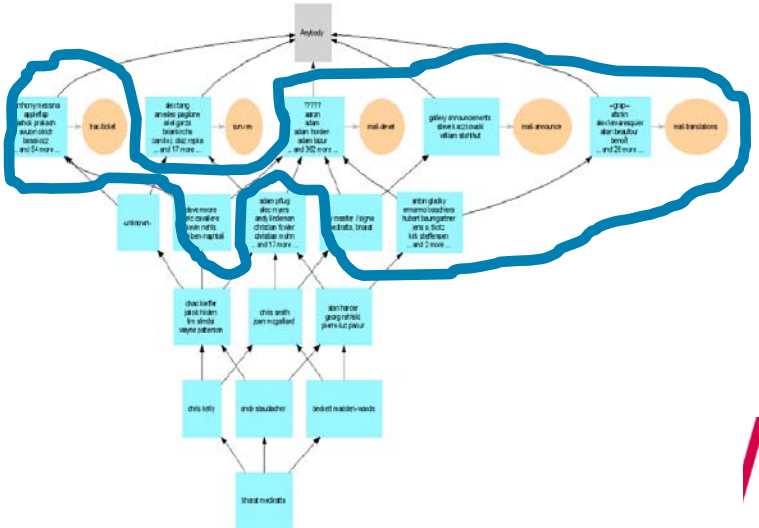
- **Development effort distribution and evolution**





- **Can be combined with other information to distinguish different kinds of developers**





# Assignment 5: Cathedral vs. Bazaar

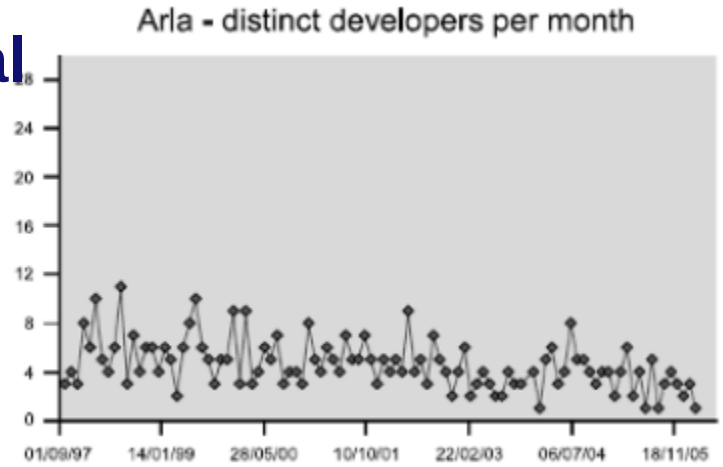
- Two Open source development models
- Raymond: two different approaches
- Capiluppi and Michlmayr: two different phases

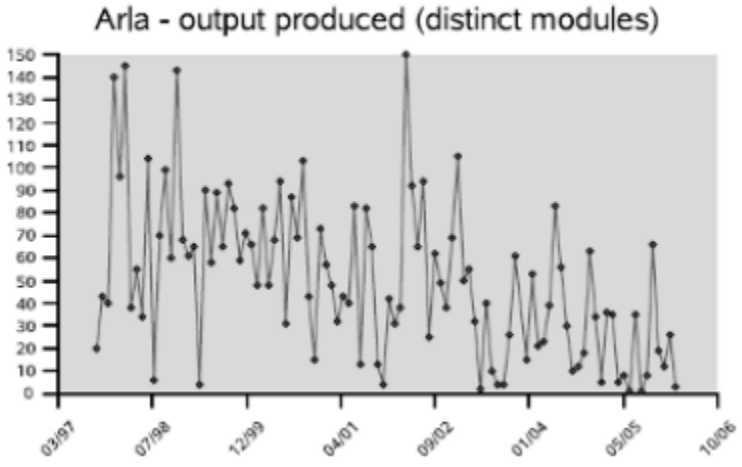
- Original idea
- Project author
- “Core developers”
- Unix philosophy

- Peer reviews
- Parallel corrective and perfective maintenance
- Community
- Distributed dev. env.

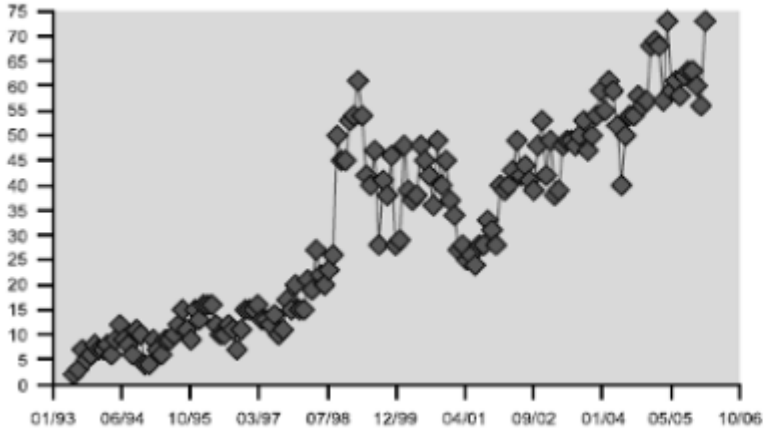
# Assignment 5: Cathedral vs. Bazaar

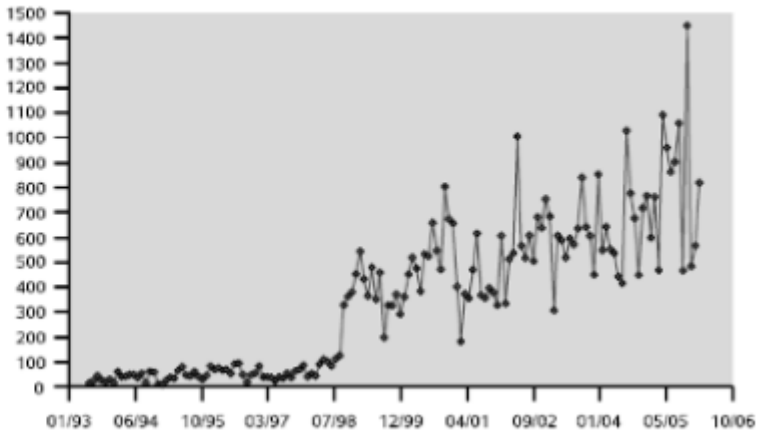
**Cathedral**




**Bazaar**



# Assignment 5: Your assignment

- **Study the evolution of the number of distinct developers per month**
- **Does it correspond to the cathedral phase or to the bazaar phase?**
- **Is your conclusion confirmed by the number of distinct files touched per month?**
- **Discuss threats to validity of your conclusions**

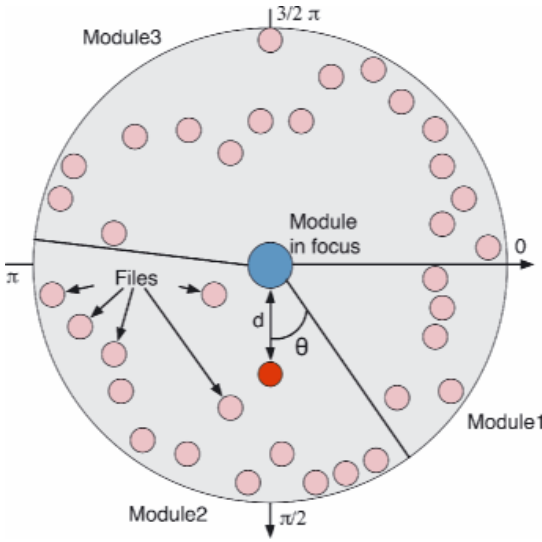
# What can we learn about files?

- **Change coupling - two artifacts change together [Ball et al. 1997]**
  - **Based on common commits**
  - **Subversion – easy, CVS – time window**
  - **Looks like EROSE**

- **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)**

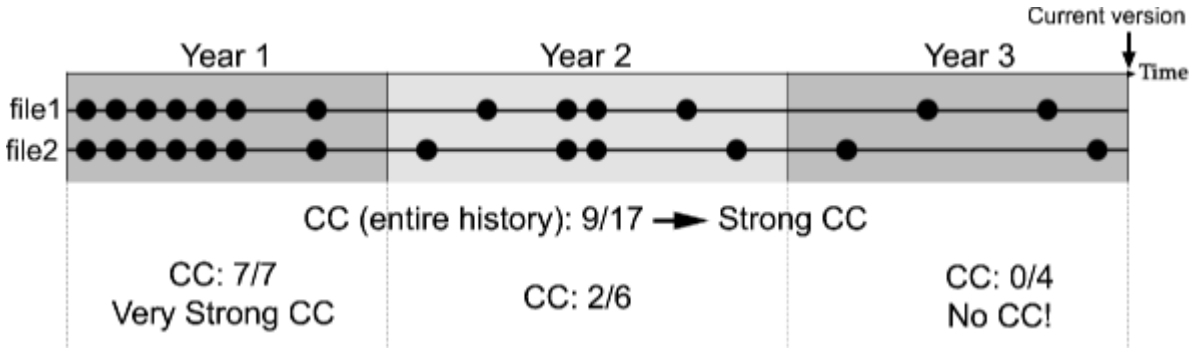
# Weapon of choice: Evolution Radar

- 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**



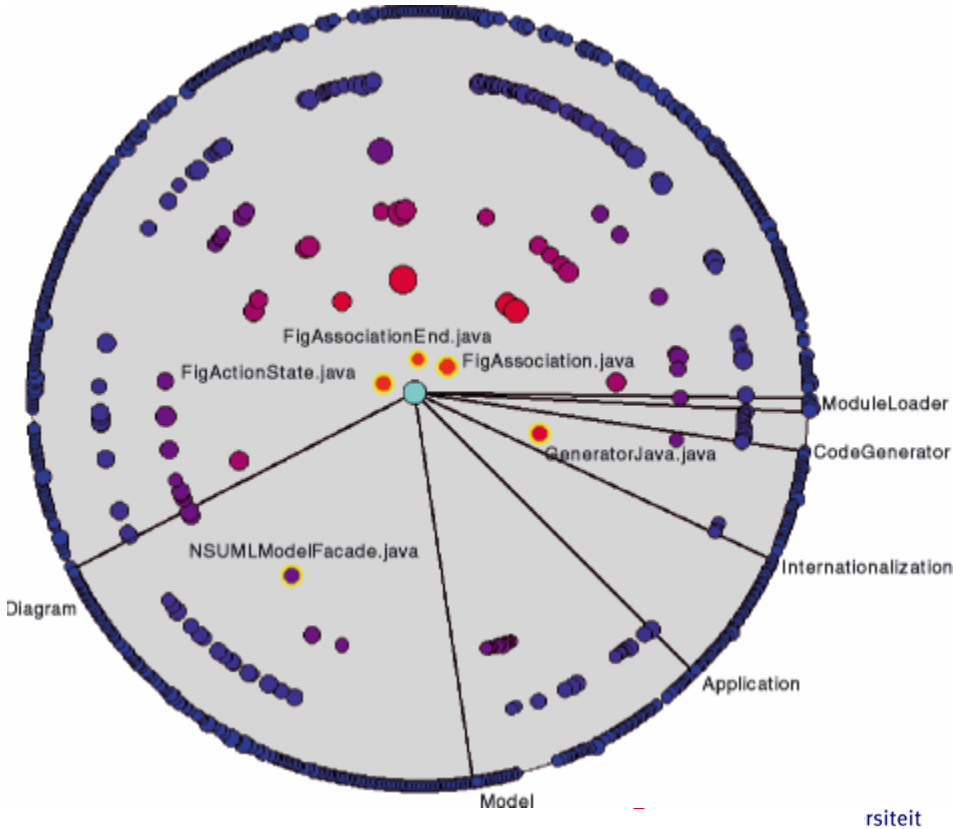

  - **Radar is time-dependent: entire history vs. time window**

- **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

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

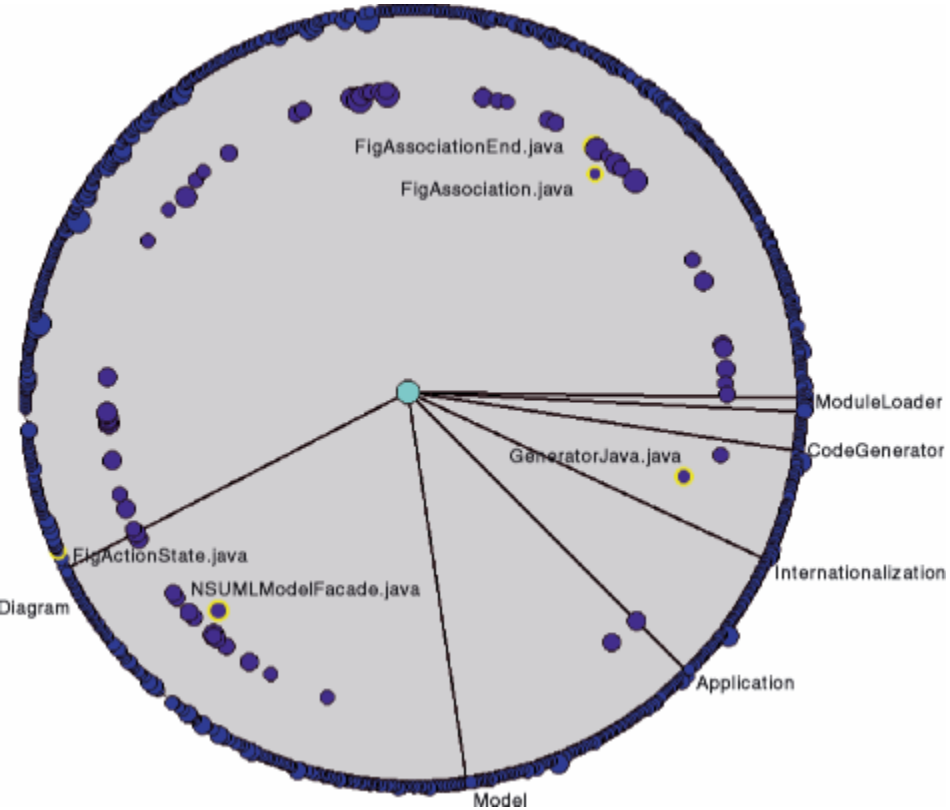



**Conclusion: Explorer strongly depends on Diagram**

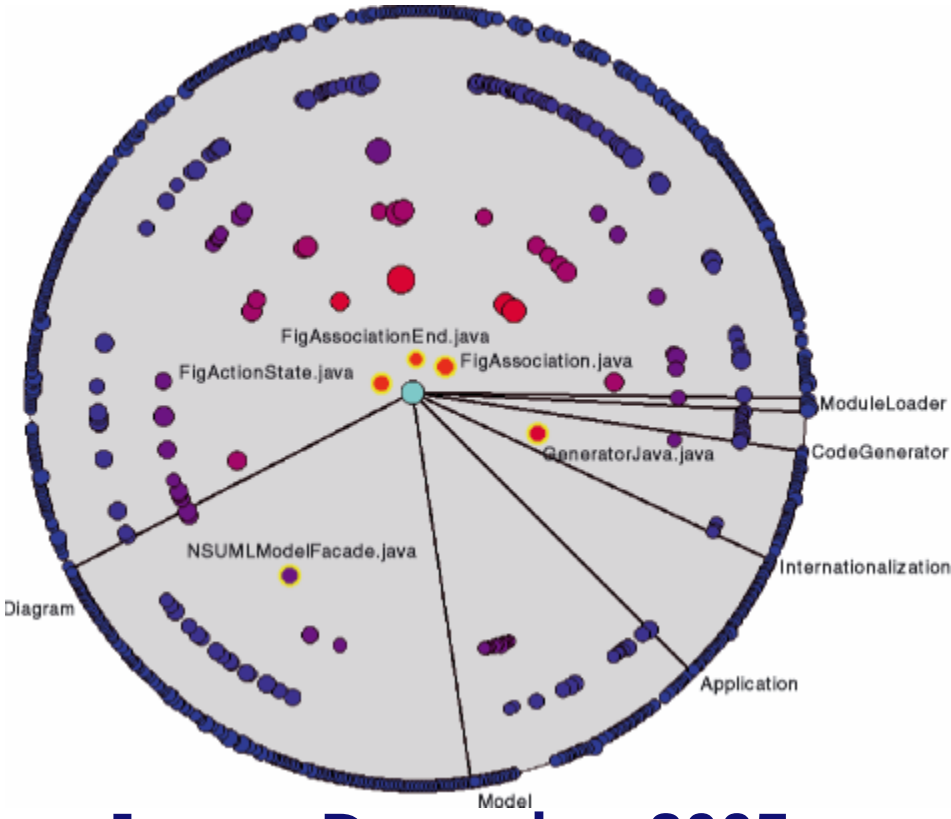
**June – December 2005**

# ArgoUML

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



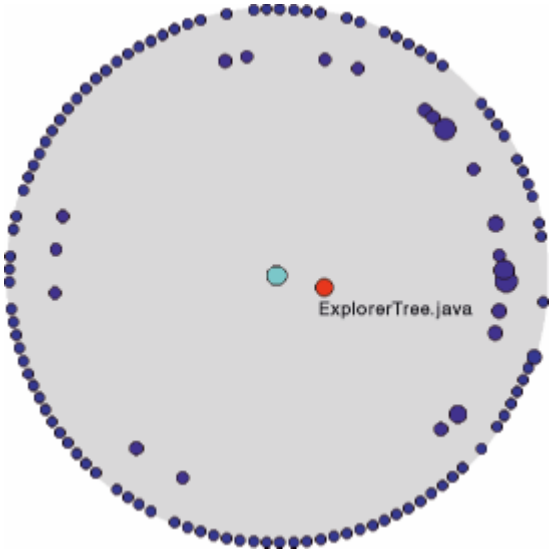

January – June 2005




June – December 2005

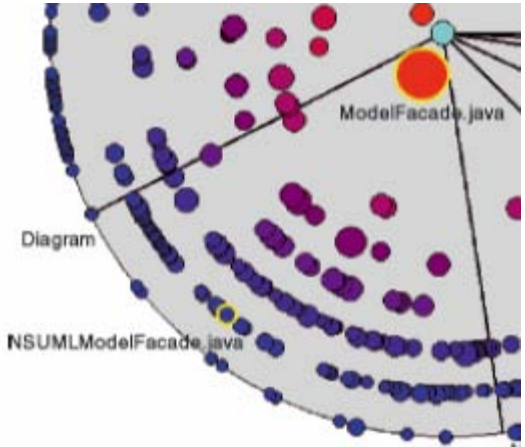
# Evolution Radar: example (cnt'd)

- **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**

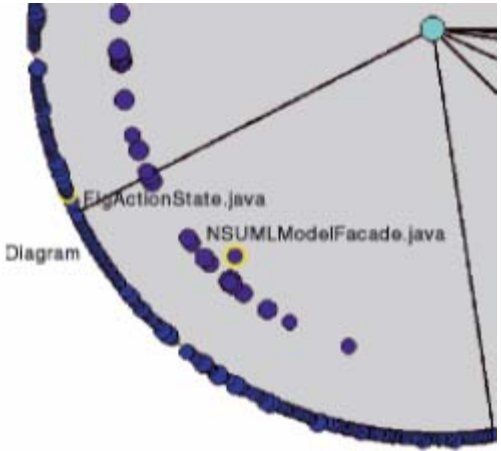



- **Problematic file was copied and removed**

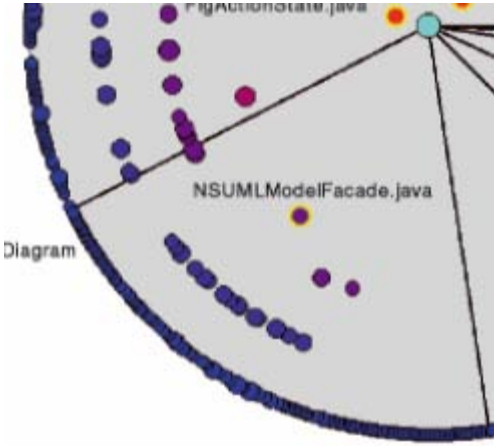
*Jun-Dec 2004*




*Jan-Jun 2004*

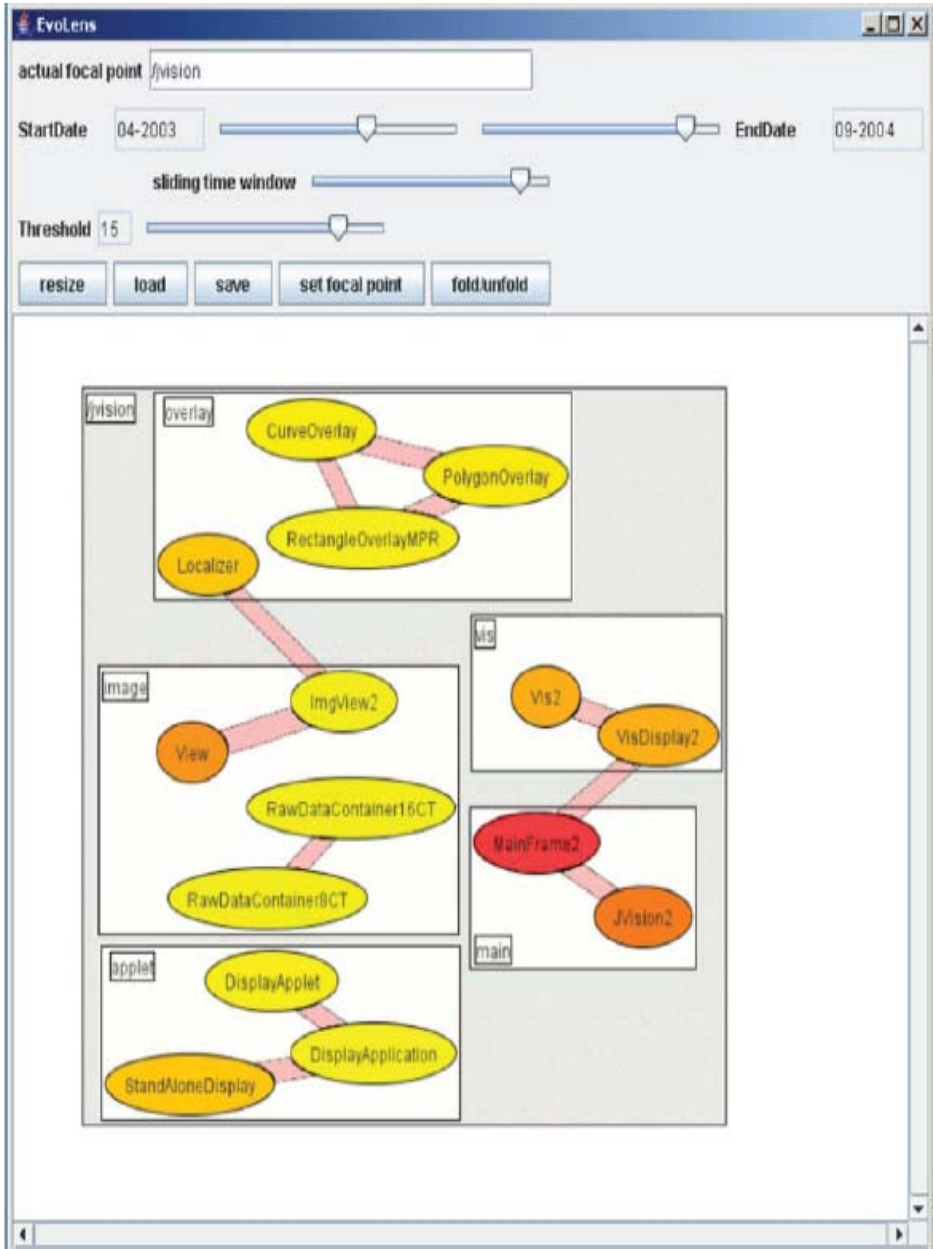



*Jun-Dec 2005*



# Alternative visualization: EvoLens

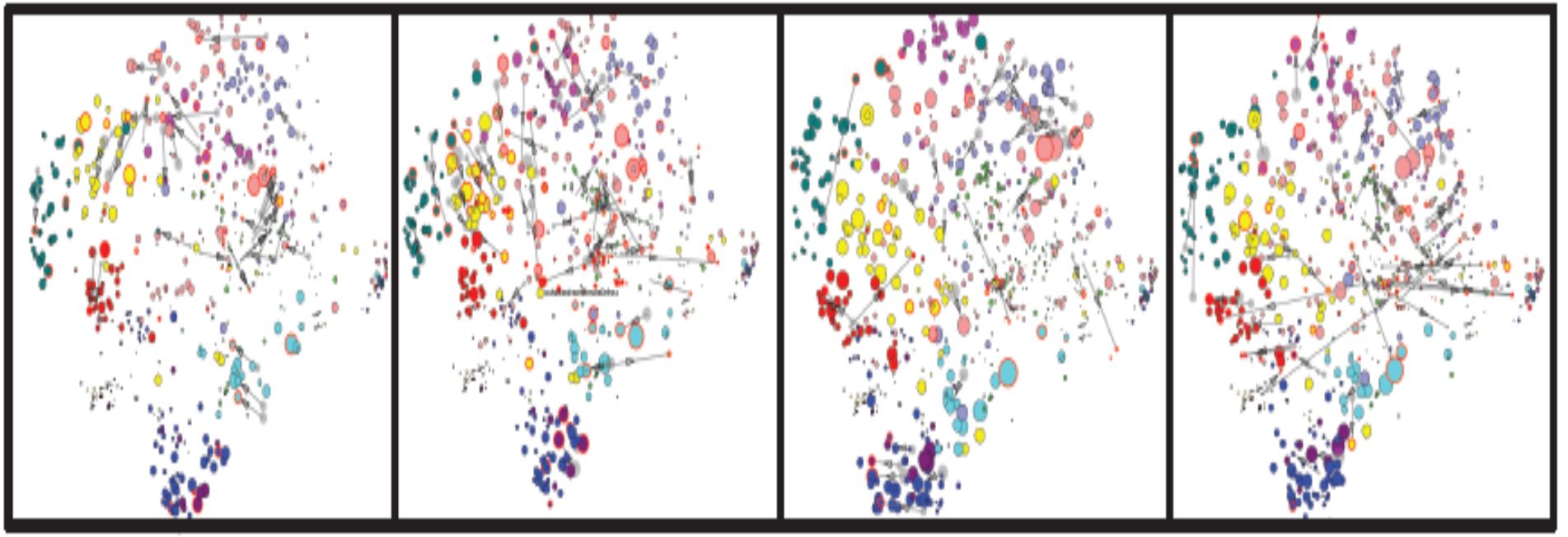
- **Focus: gray rectangle**
- **2 hierarchy levels: classes are “flattened” to submodules**
- **Colours: growth speed**
- **Edges: strength of change coupling**
- **[Ratzinger, Fischer, Gall, 2005]**



# 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




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

# Conclusions

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

- **Useful in combination with extra information about**
  - **code size (e.g., fractal figures)**
  - **dependencies (e.g., evolution storyboards)**
  - **inheritance (e.g., Collberg et al.)**