UNIVERSITÄT BERN

# Software Ecosystem Analysis

Mircea Lungu

Software Design and Evolution, December 7, 2011

# Software is Data...

- Data that you analyze
- Data that you measure
- Data that evolves and can be *mined*
- Executable data
- **... big data**
- Data that you visualize

# Roadmap

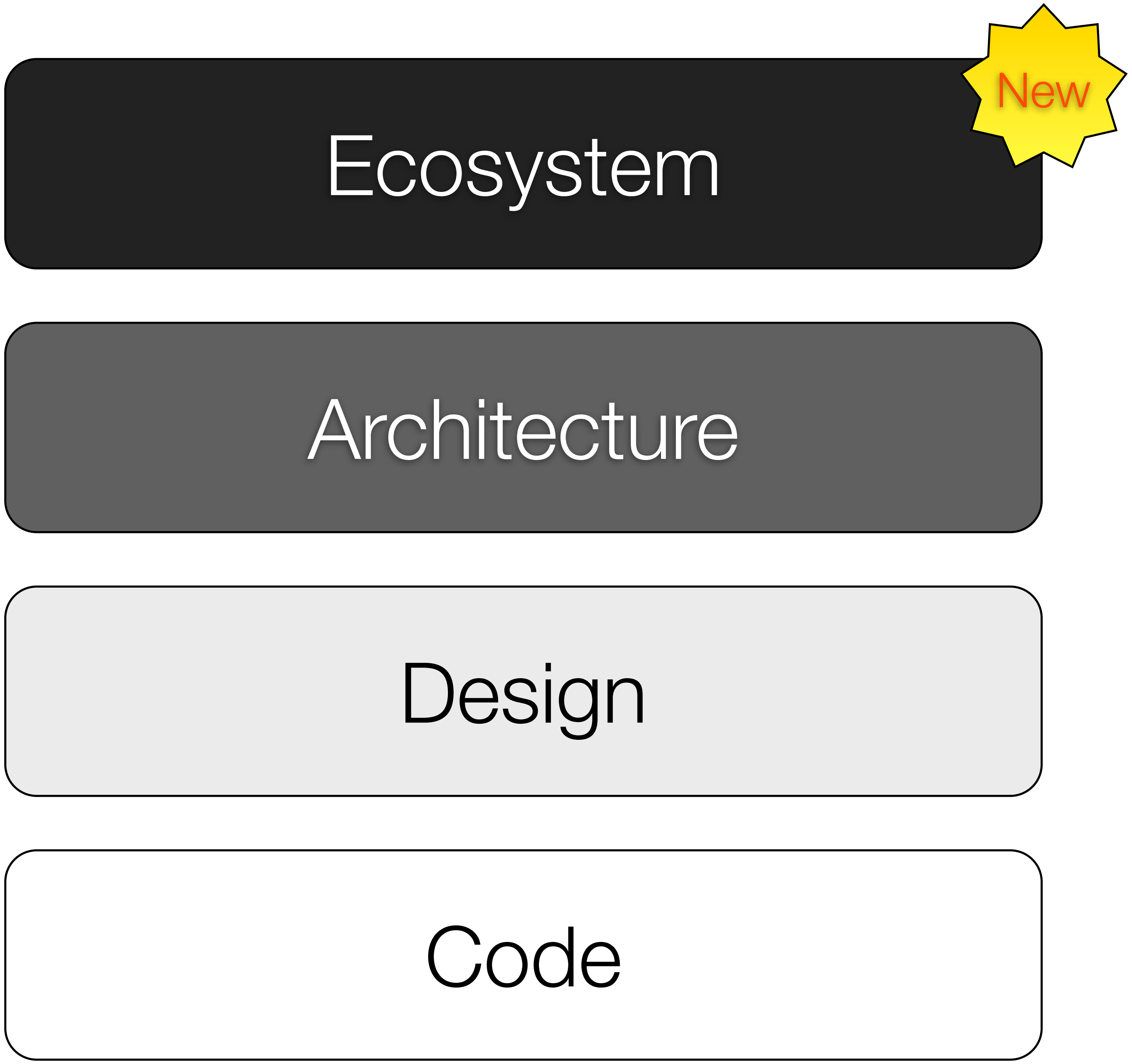
- Software Ecosystems
- Reverse Engineering Software Ecosystems
- Dependency Analysis
- API Evolution
- And more...

# Main Materials

- **Recovering Inter-Project Dependencies in Software Ecosystems**, Lungu & Robbes, 2010
- **Automated Dependency Resolution for Open Source Software**, Ossher et al., 2010
- **A Study of Ripple Effects in Software Ecosystems**, Robbes & Lungu, 2011
- **Mining Framework Usage Changes from Instantiation code**, Schaeffer et al. 2008
- **File Cloning in Open Source Systems: The Good, The Bad and The Ugly**, Ossher et al. 2011

# Roadmap

- **Software Ecosystems**
- Reverse Engineering Software Ecosystems
- Dependency Analysis
- API Evolution
- And more...



New

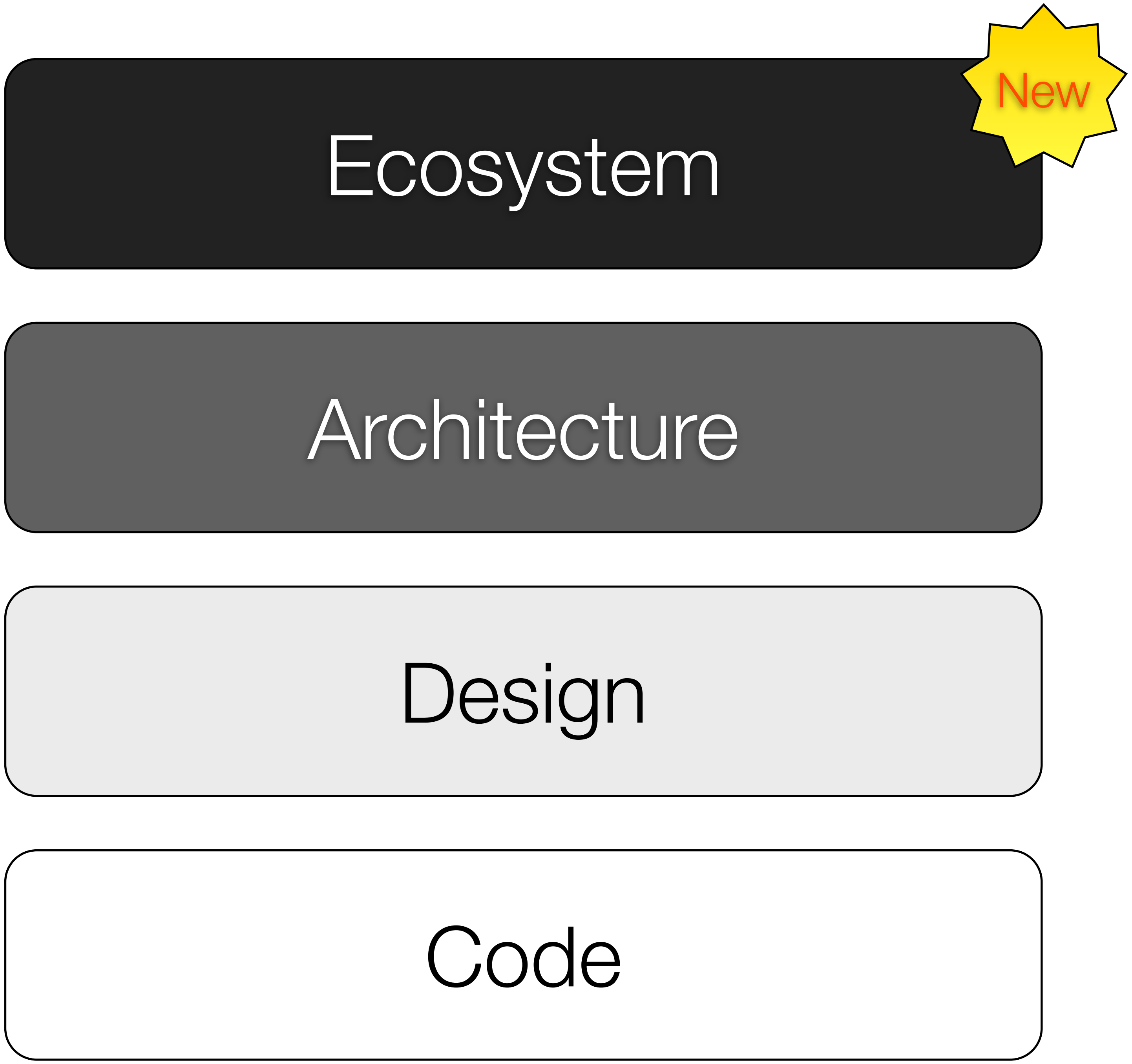
Ecosystem

Architecture

Design

Code

# The ecosystems is an abstraction level for software that is *above* the architecture



# Definition

- A software ecosystem is a collection of software systems which are developed and which co-evolve together in the same environment.

[Lungu '09]

# An environment can be many things

# An environment can be many things

> An open source community

# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project

# An environment can be many things

> An open source community
>   — Apache Software Foundation, The Gnome Project
> A company

# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project
- A company
  - <your large company here>

# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project
- A company
  - <your large company here>
- An academic institution

# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project
- A company
  - <your large company here>
- An academic institution
  - The SCG Research Group

# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project
- A company
  - <your large company here>
- An academic institution
  - The SCG Research Group
- A programming language

# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project
- A company
  - <your large company here>
- An academic institution
  - The SCG Research Group
- A programming language
  - All the software written in Perl, Smalltalk, etc.

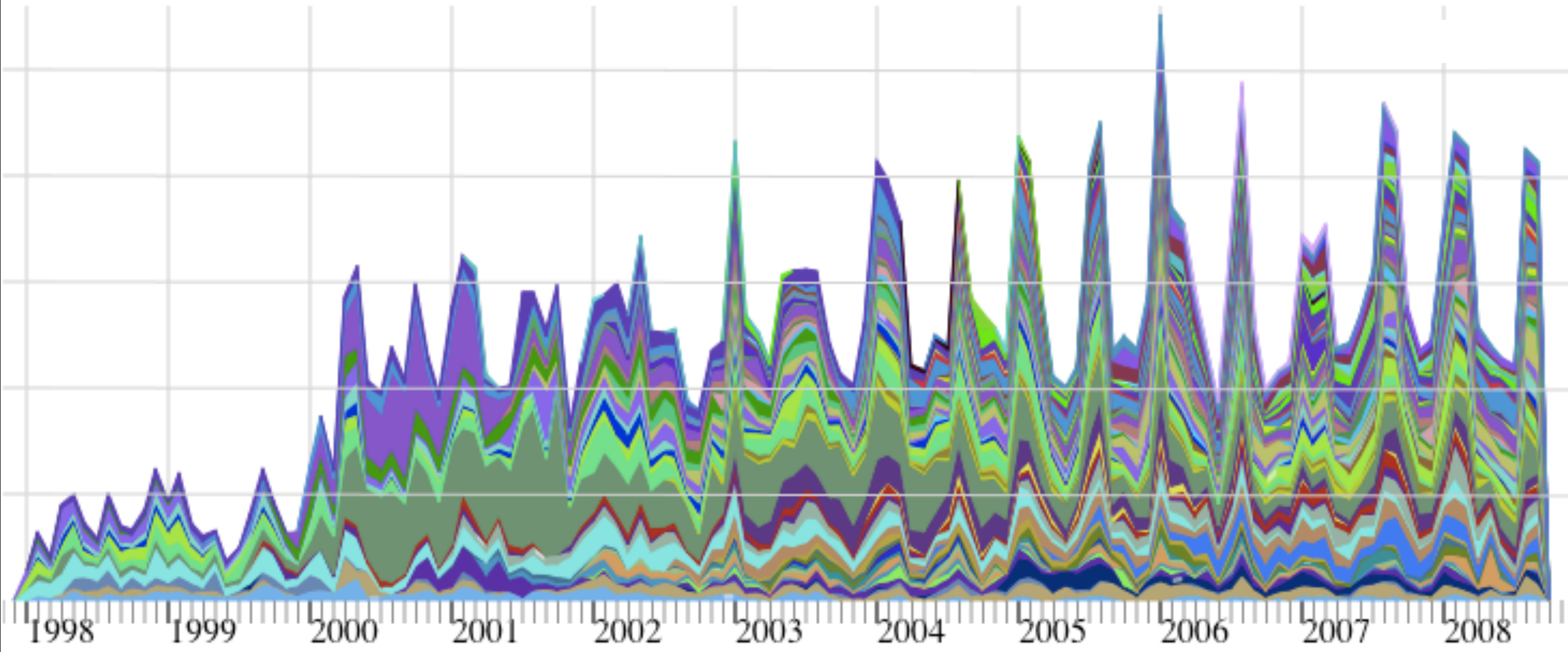
# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project
- A company
  - <your large company here>
- An academic institution
  - The SCG Research Group
- A programming language
  - All the software written in Perl, Smalltalk, etc.
- A technology/platform

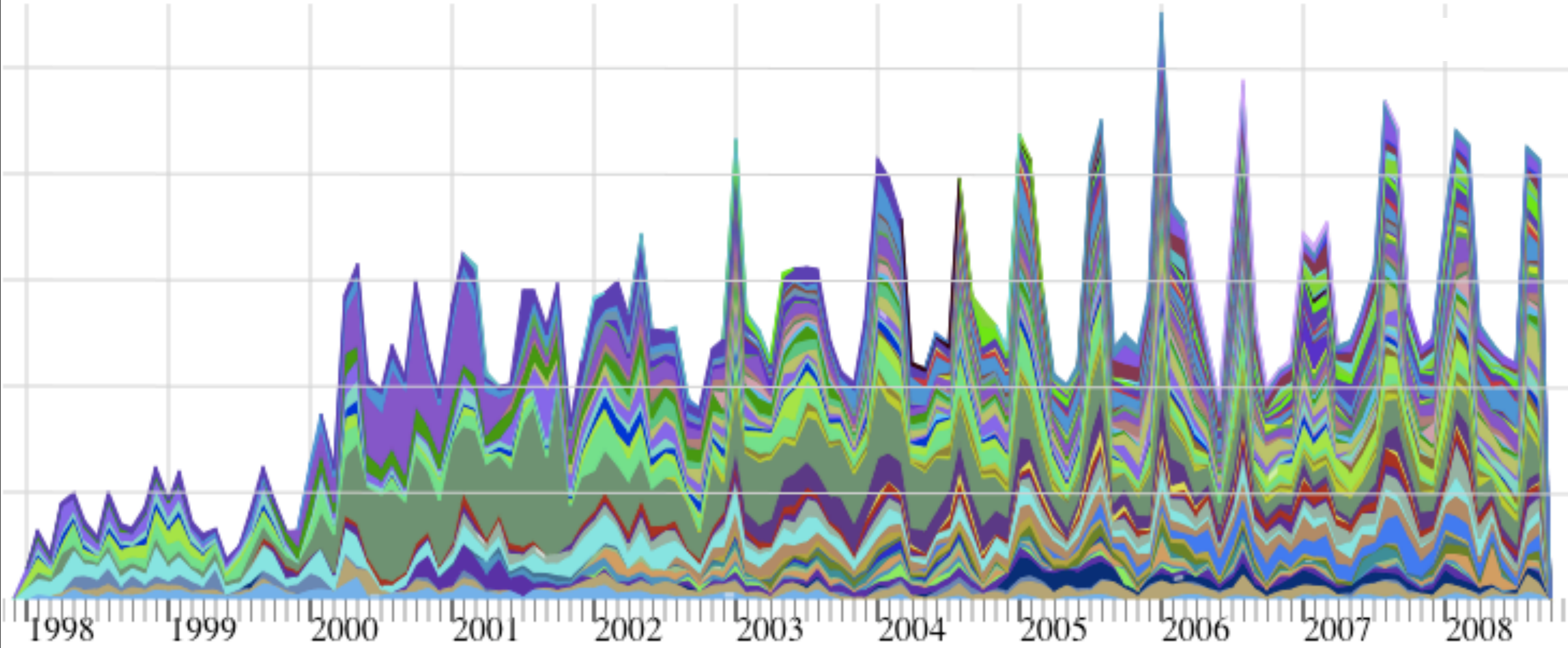
# An environment can be many things

- An open source community
  - Apache Software Foundation, The Gnome Project
- A company
  - <your large company here>
- An academic institution
  - The SCG Research Group
- A programming language
  - All the software written in Perl, Smalltalk, etc.
- A technology/platform
  - Eclipse

# An Example: The Evolution of Gnome

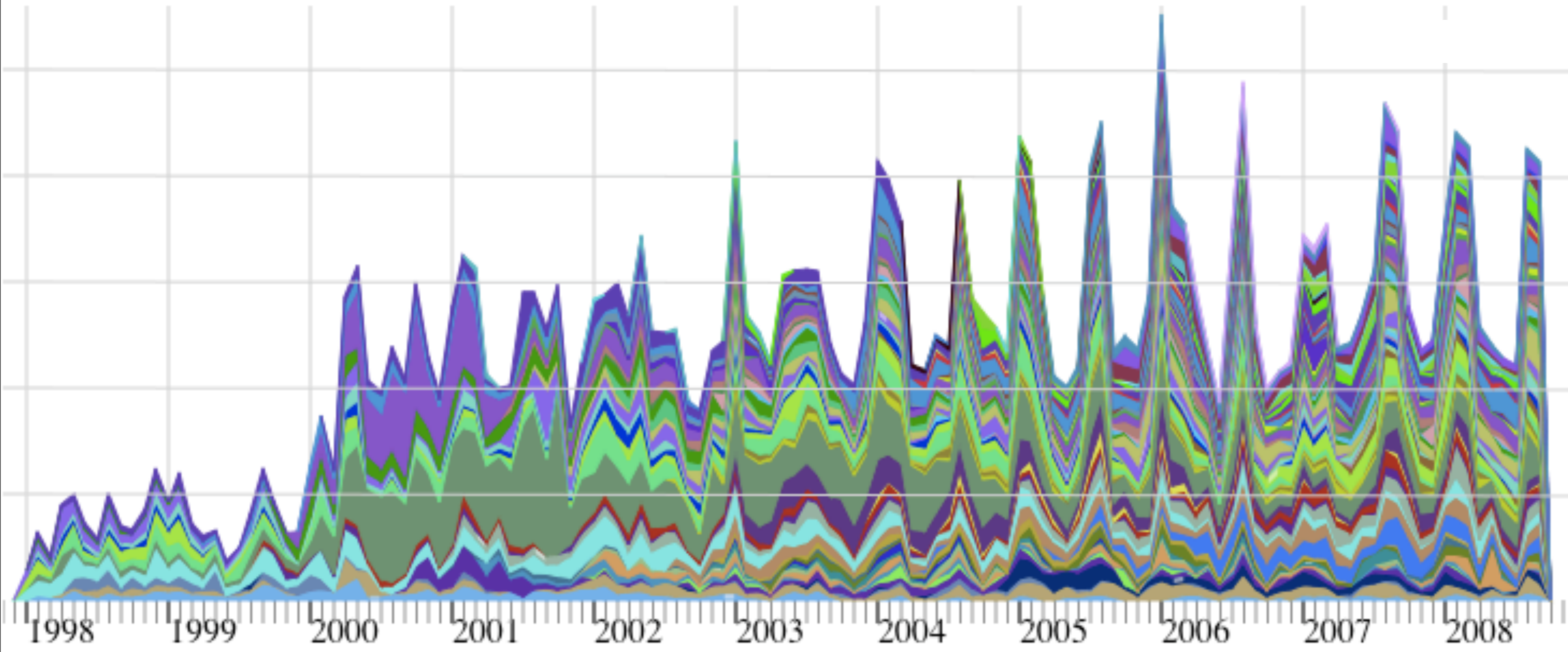


# An Example: The Evolution of Gnome



Introduction

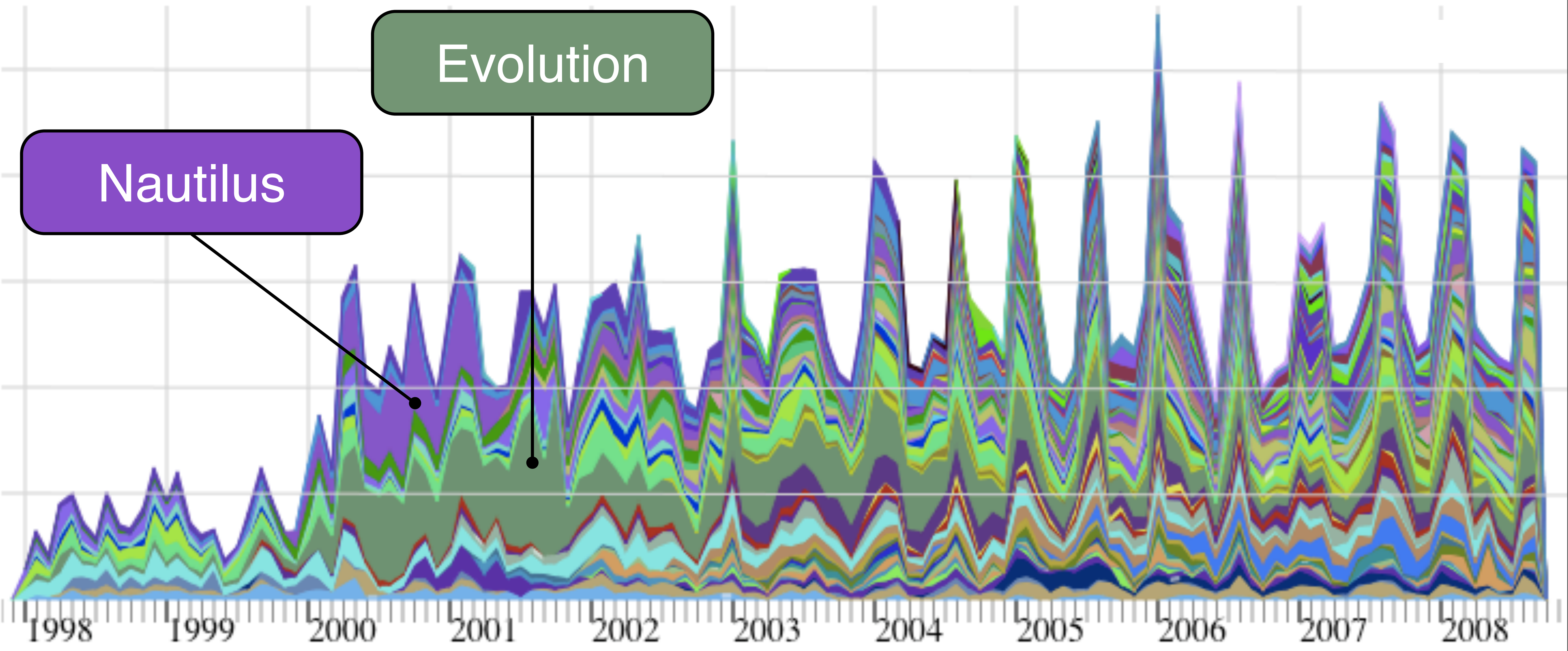
# An Example: The Evolution of Gnome



Introduction

Growth

# An Example: The Evolution of Gnome



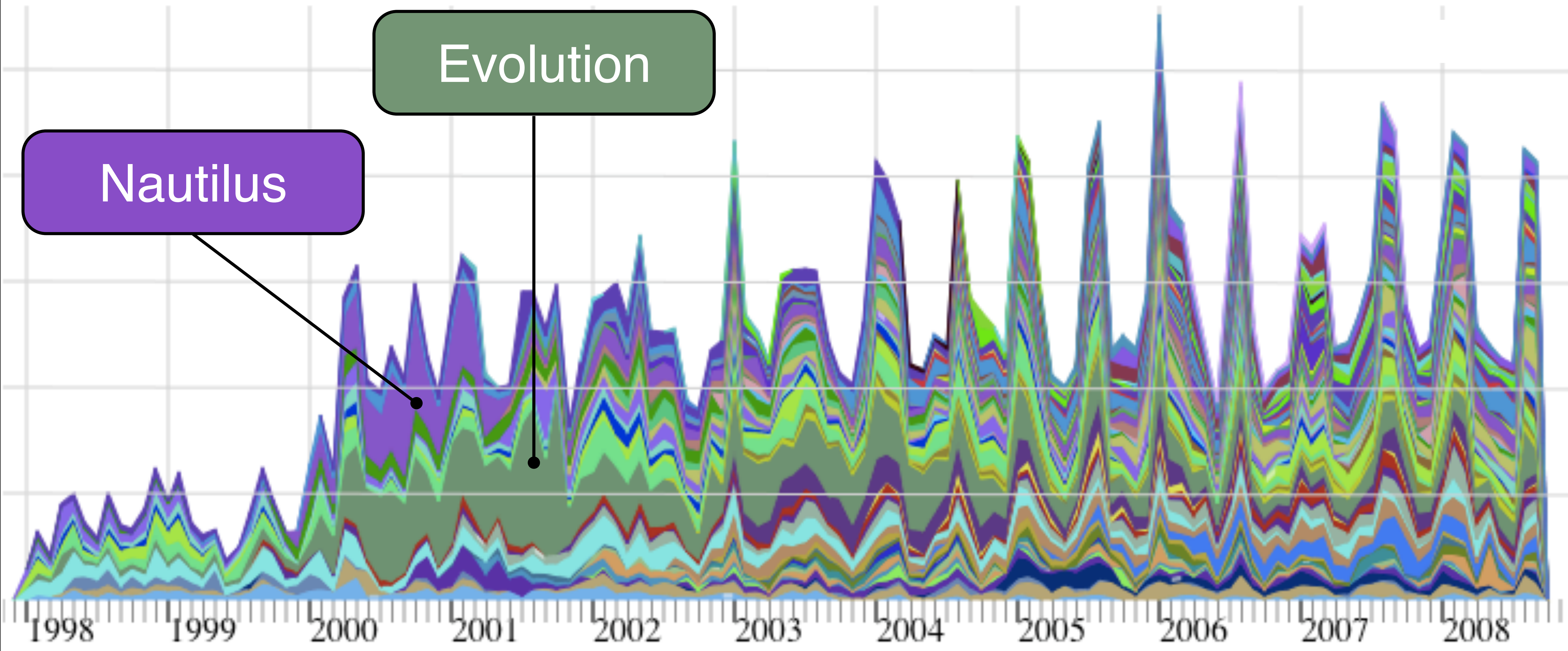
Evolution

Nautilus

Introduction

Growth

# An Example: The Evolution of Gnome



Evolution

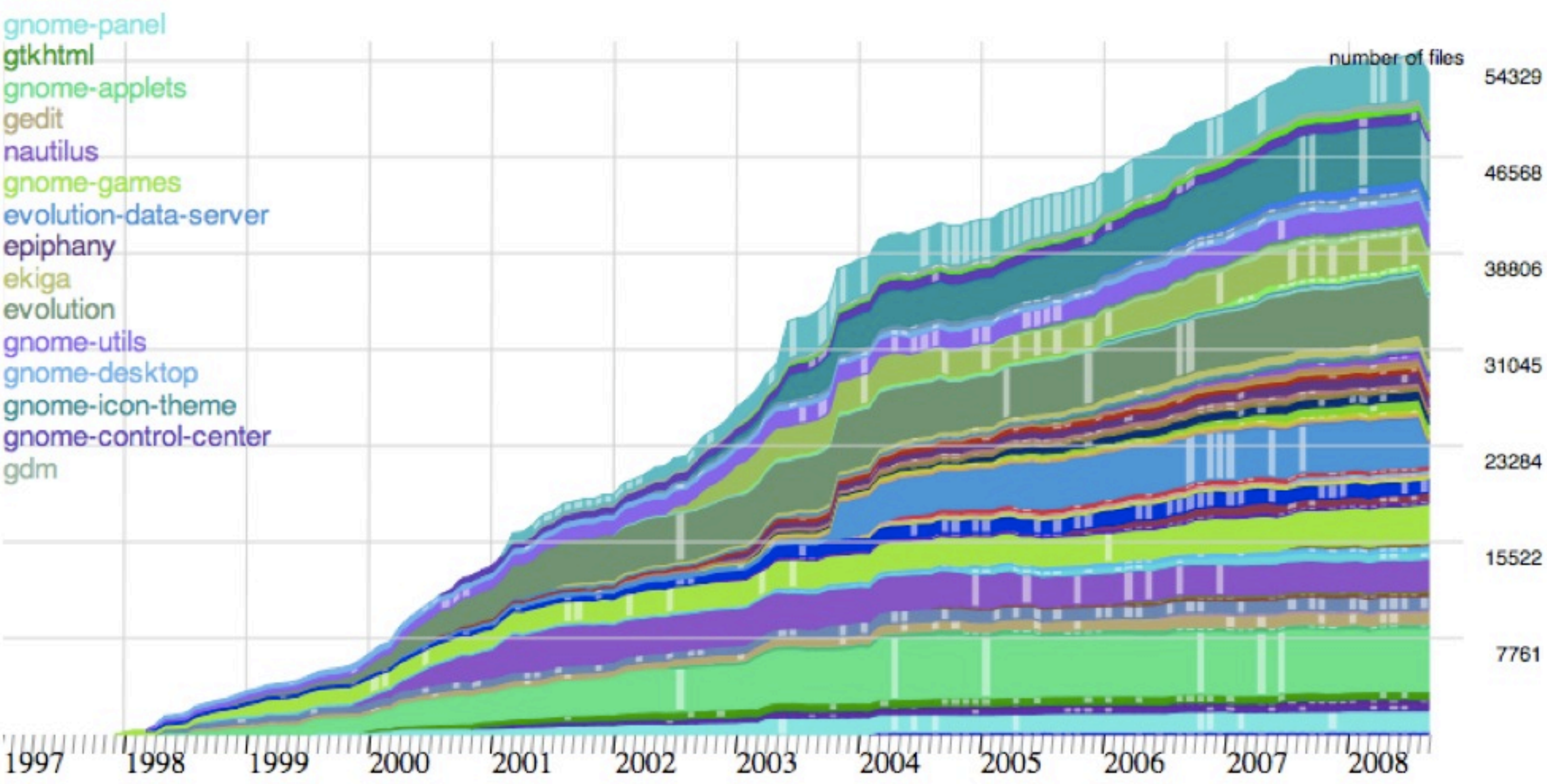
Nautilus

Introduction

Growth

Maturity

# Size Evolution - Gnome



# Super-Repositories

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem
- Related with a software ecosystem

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem
- Related with a software ecosystem
  - 1-to-1

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem

- Related with a software ecosystem
  - 1-to-1
  - 1-to-many

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem

- Related with a software ecosystem
  - 1-to-1
  - 1-to-many

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem
- Related with a software ecosystem
  - 1-to-1
  - 1-to-many
- Two types

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem
- Related with a software ecosystem
  - 1-to-1
  - 1-to-many
- Two types
  - Language-based

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem

- Related with a software ecosystem
  - 1-to-1
  - 1-to-many

- Two types
  - Language-based
    - *SqueakSource, RubyForge, CPAN*

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem

- Related with a software ecosystem
  - 1-to-1
  - 1-to-many

- Two types
  - Language-based
    - *SqueakSource, RubyForge, CPAN*
  - Language-Agnostic

# Super-Repositories

- A super-repository is the collection of all the versioning control repositories for the systems that are part of an ecosystem

- Related with a software ecosystem
  - 1-to-1
  - 1-to-many

- Two types
  - Language-based
    - *SqueakSource, RubyForge, CPAN*
  - Language-Agnostic
    - *GitHub, SourceForge, <your company's folder with svn repos>*

# Similar Concepts

# Similar Concepts

- Product families
  - Several systems that are very similar
  - Core architecture with variations

# Similar Concepts

- Product families
  - Several systems that are very similar
  - Core architecture with variations
- Business ecosystems
  - a set of businesses functioning as a unit and interacting with a shared market for software and services (Szyperski, 2003)
  - not interesting for us

# Similar Concepts

- Product families
  - Several systems that are very similar
  - Core architecture with variations
- Business ecosystems
  - a set of businesses functioning as a unit and interacting with a shared market for software and services (Szyperski, 2003)
  - not interesting for us
- Individual systems ...
  - if we look at the code level it might be the same thing
  - two “p”’s are different
    - *properties*
    - *problems*

# Roadmap

- Software Ecosystems
- **Reverse Engineering Software Ecosystems**
- Dependency Analysis
- API Evolution
- And more...

# RevEngE: An approach to understanding an ecosystem

> Lungu ’09

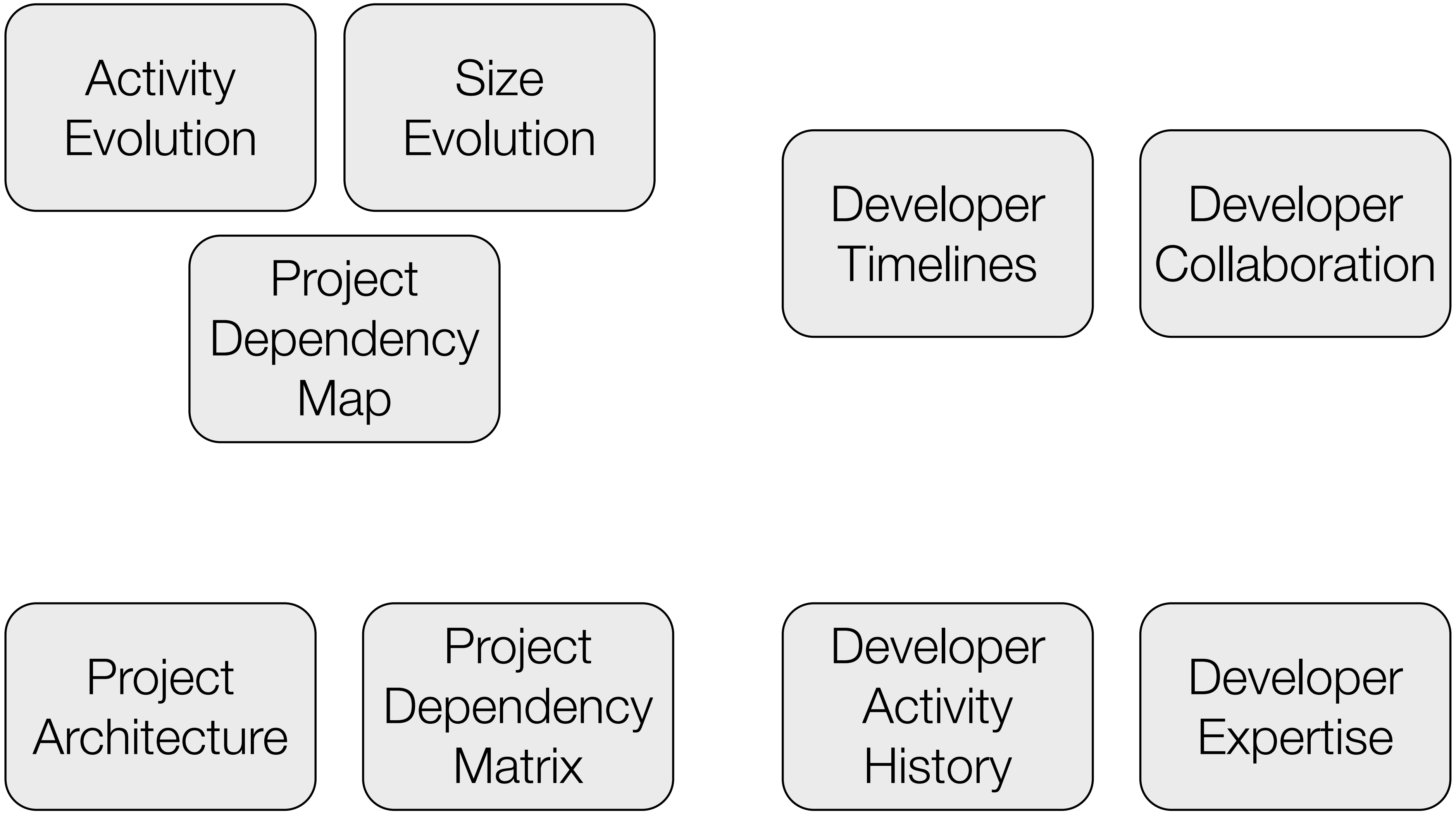
> Goal: Understanding the ecosystem as a whole and the component systems in context

# Approach

- Goal: Recover ecosystem viewpoints
- An **ecosystem viewpoint** - is a perspective on an ecosystem that presents a specific aspect of the ecosystem in order to support one or more concerns about the ecosystem.

# Ecosystem Viewpoints

# Ecosystem Viewpoints



Activity Evolution

Size Evolution

Project Dependency Map

Developer Timelines

Developer Collaboration

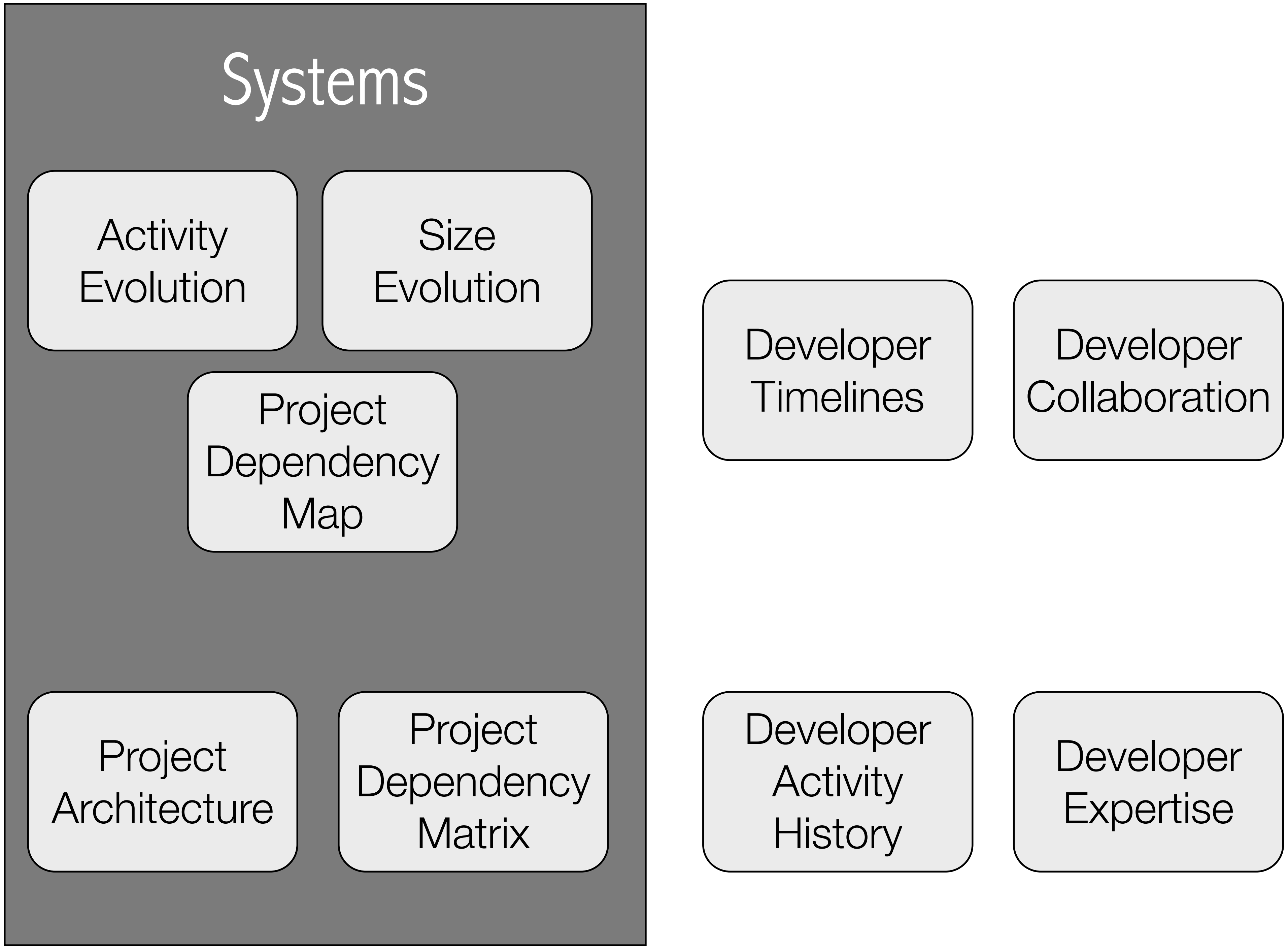
Project Architecture

Project Dependency Matrix

Developer Activity History

Developer Expertise

# Ecosystem Viewpoints



Systems

Activity Evolution

Size Evolution

Project Dependency Map

Project Architecture

Project Dependency Matrix

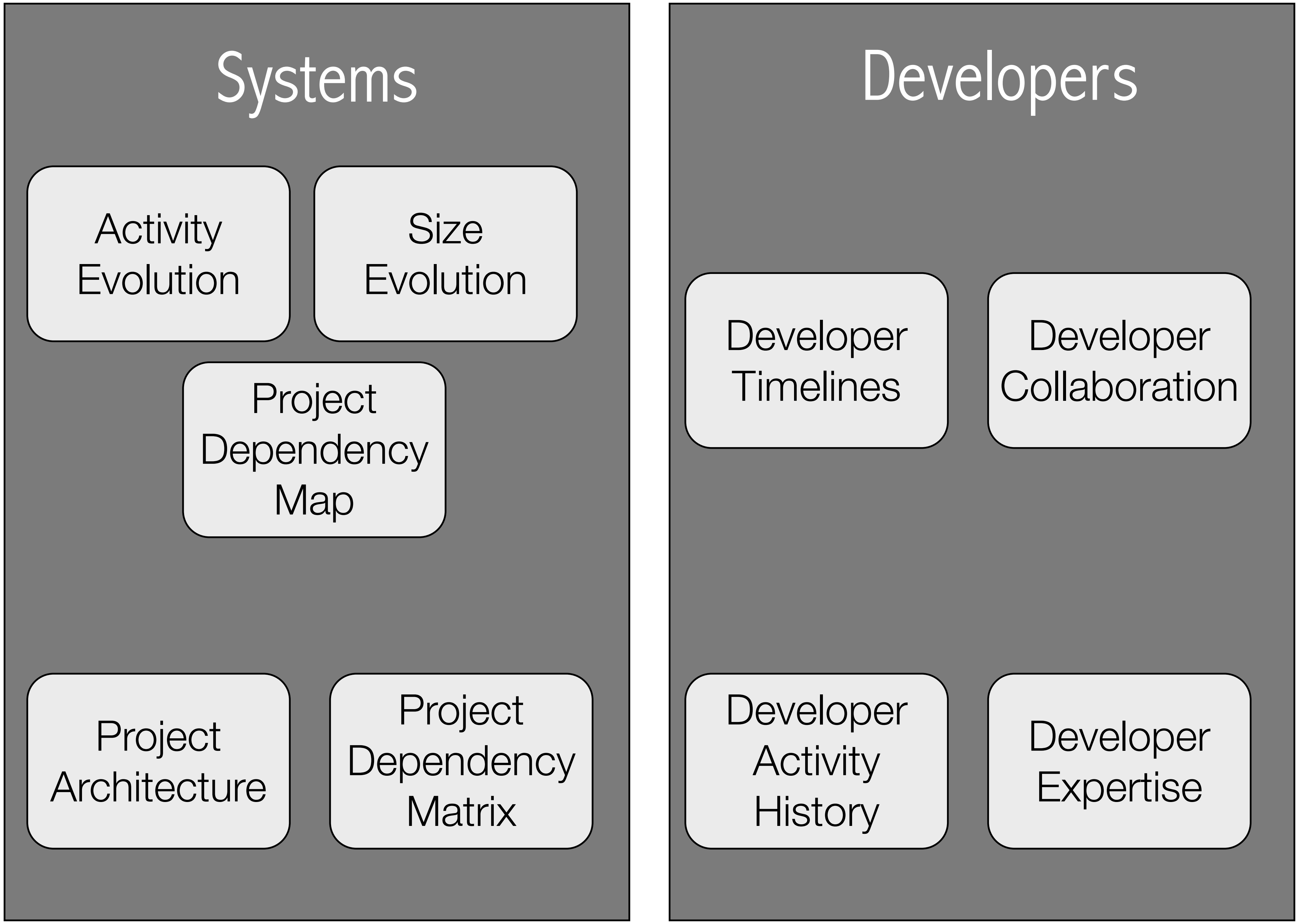
Developer Timelines

Developer Collaboration

Developer Activity History

Developer Expertise

# Ecosystem Viewpoints



Systems

Activity Evolution

Size Evolution

Project Dependency Map

Project Architecture

Project Dependency Matrix

Developers

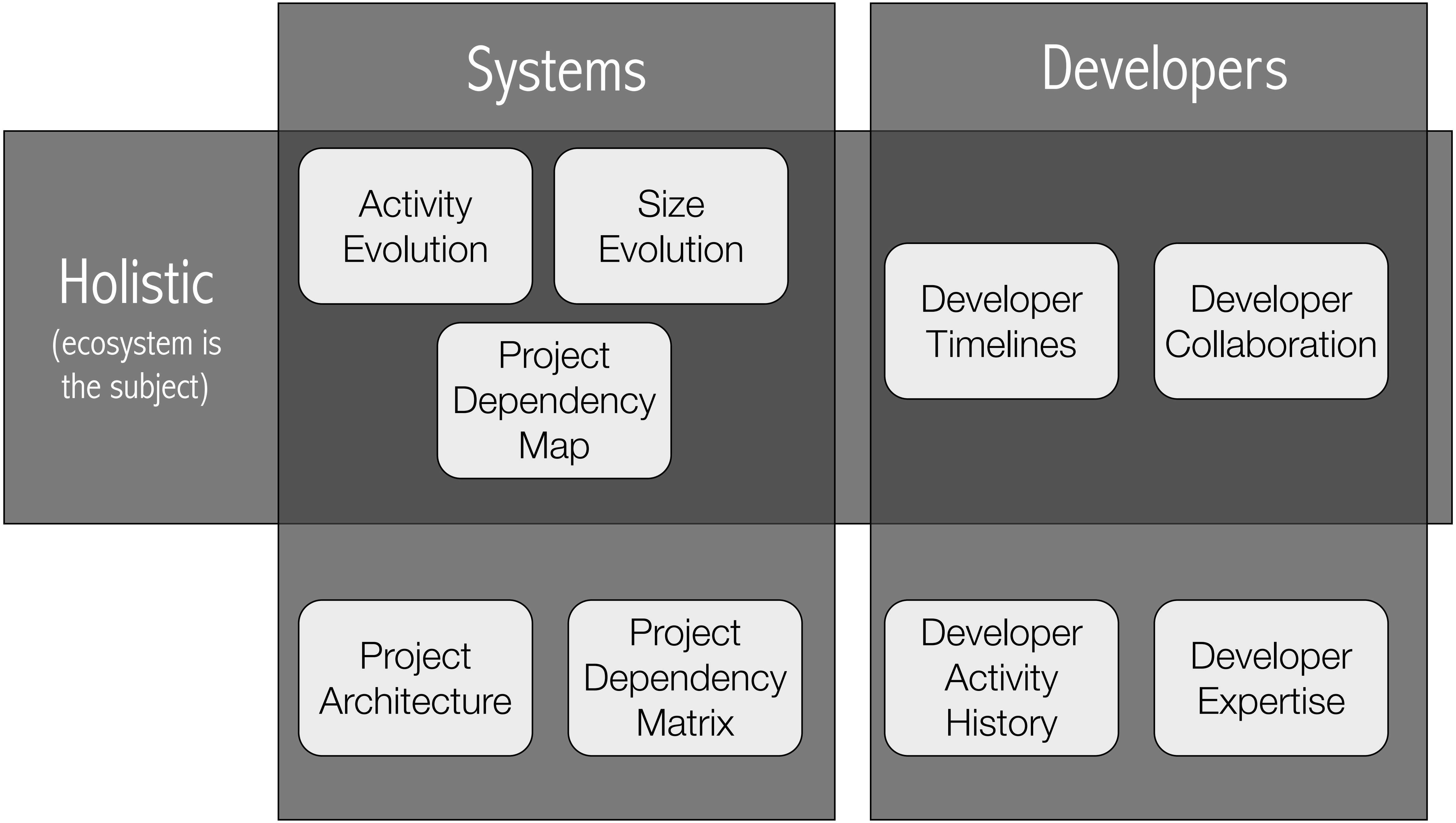
Developer Timelines

Developer Collaboration

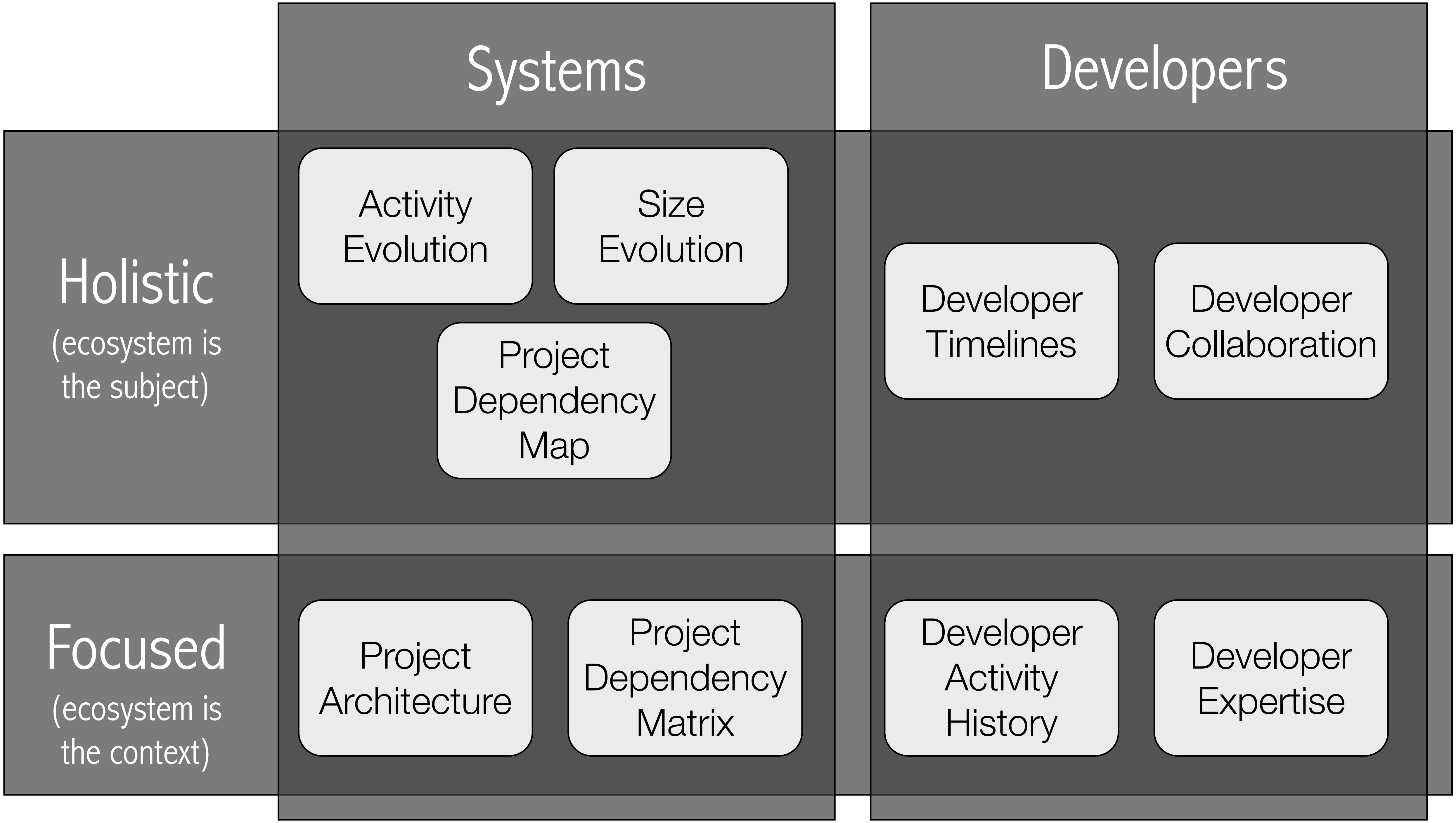
Developer Activity History

Developer Expertise

# Ecosystem Viewpoints

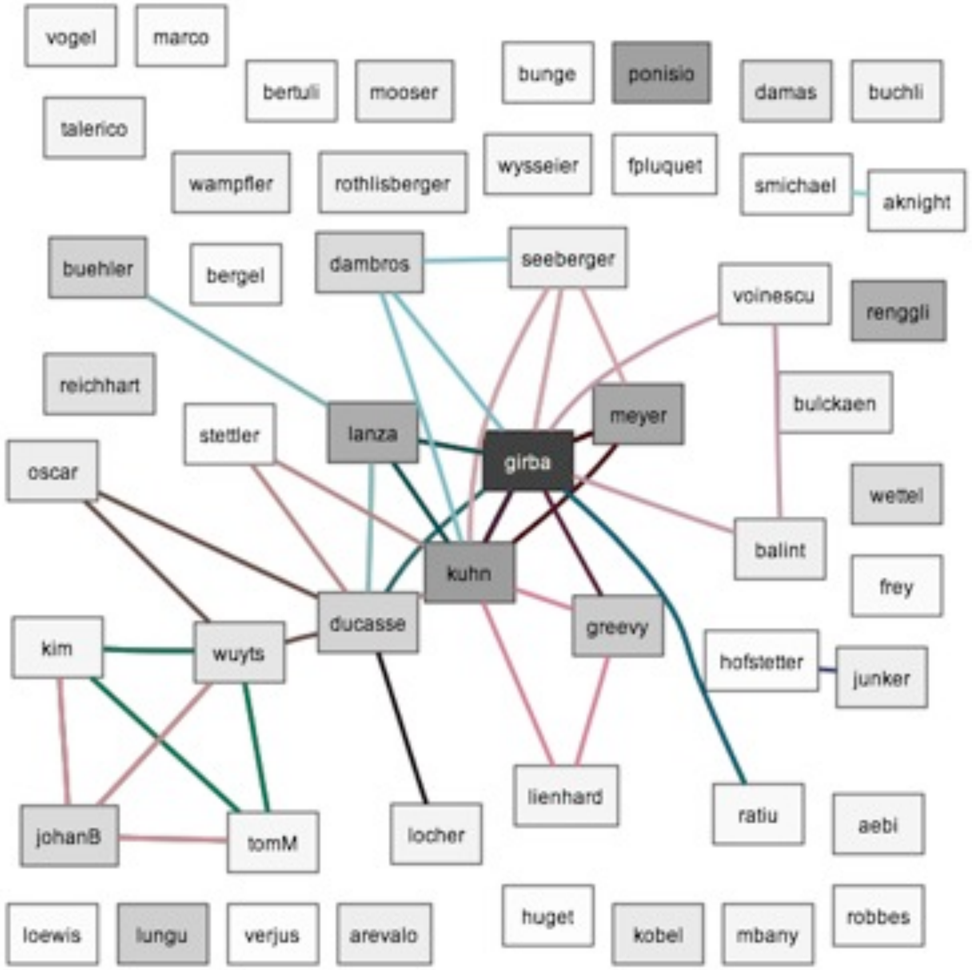


# Ecosystem Viewpoints

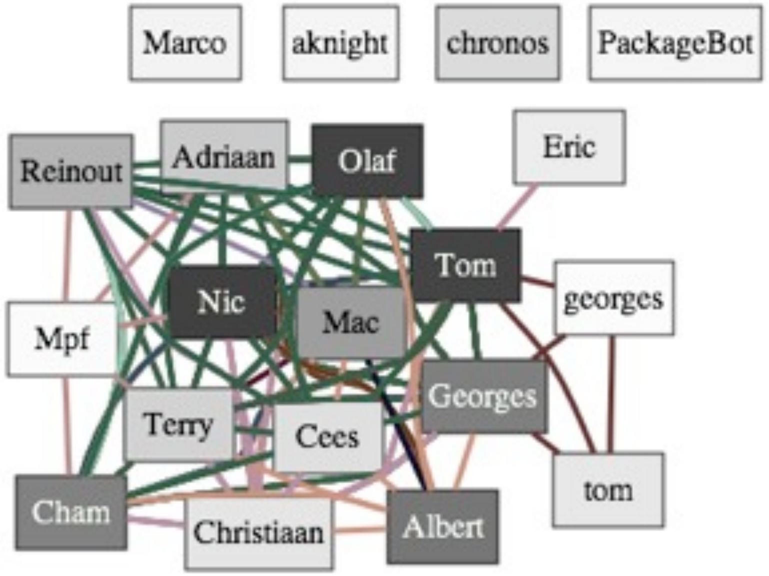


| | Systems | Developers |
|---|---|---|
| Holistic<br>(ecosystem is the subject) | Activity Evolution<br>Size Evolution<br>Project Dependency Map | Developer Timelines<br>Developer Collaboration |
| Focused<br>(ecosystem is the context) | Project Architecture<br>Project Dependency Matrix | Developer Activity History<br>Developer Expertise |

# Developer Collaboration Viewpoint

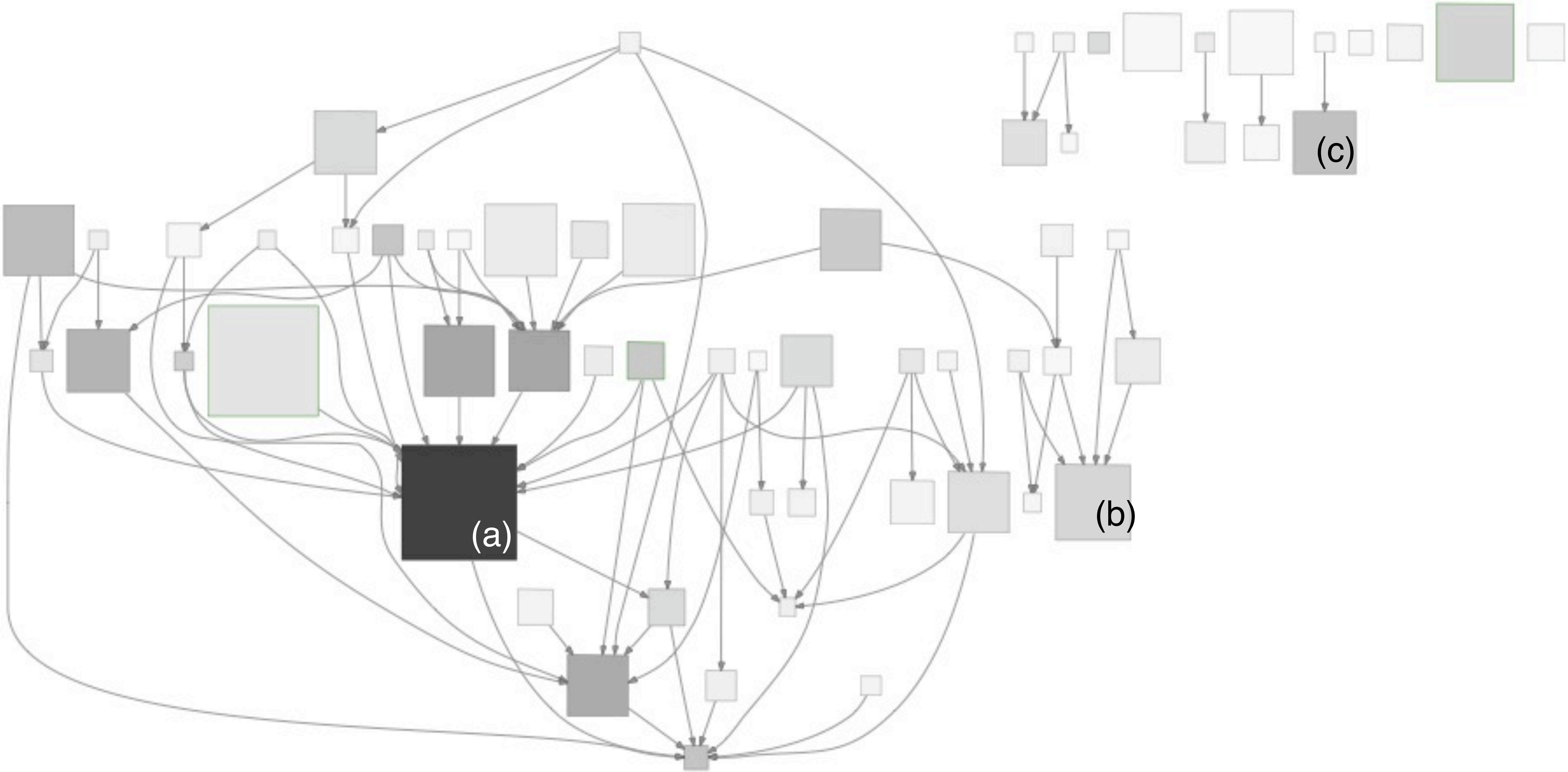


SCG 2007



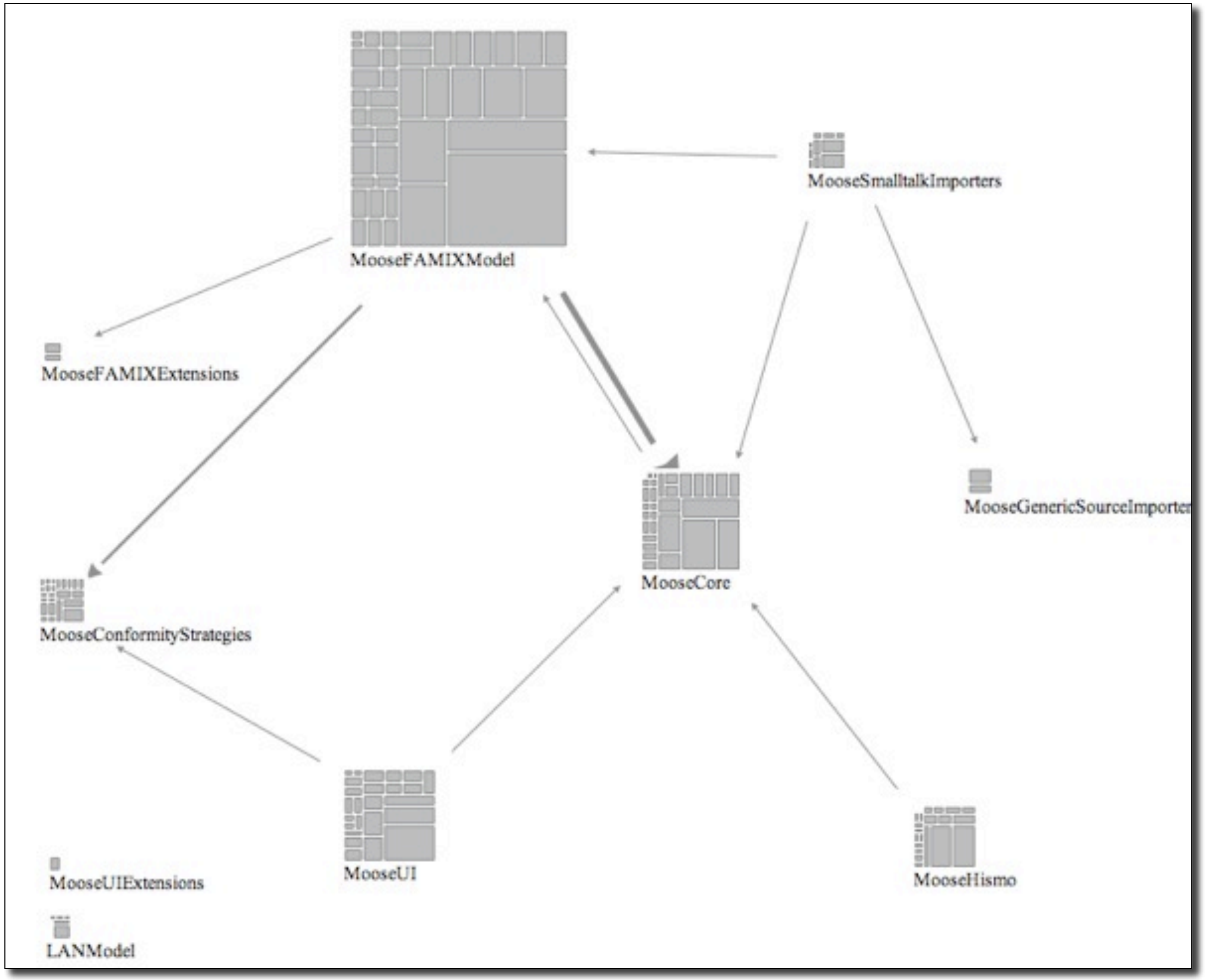
Soops 2007

# Inter-Project Dependencies Viewpoint



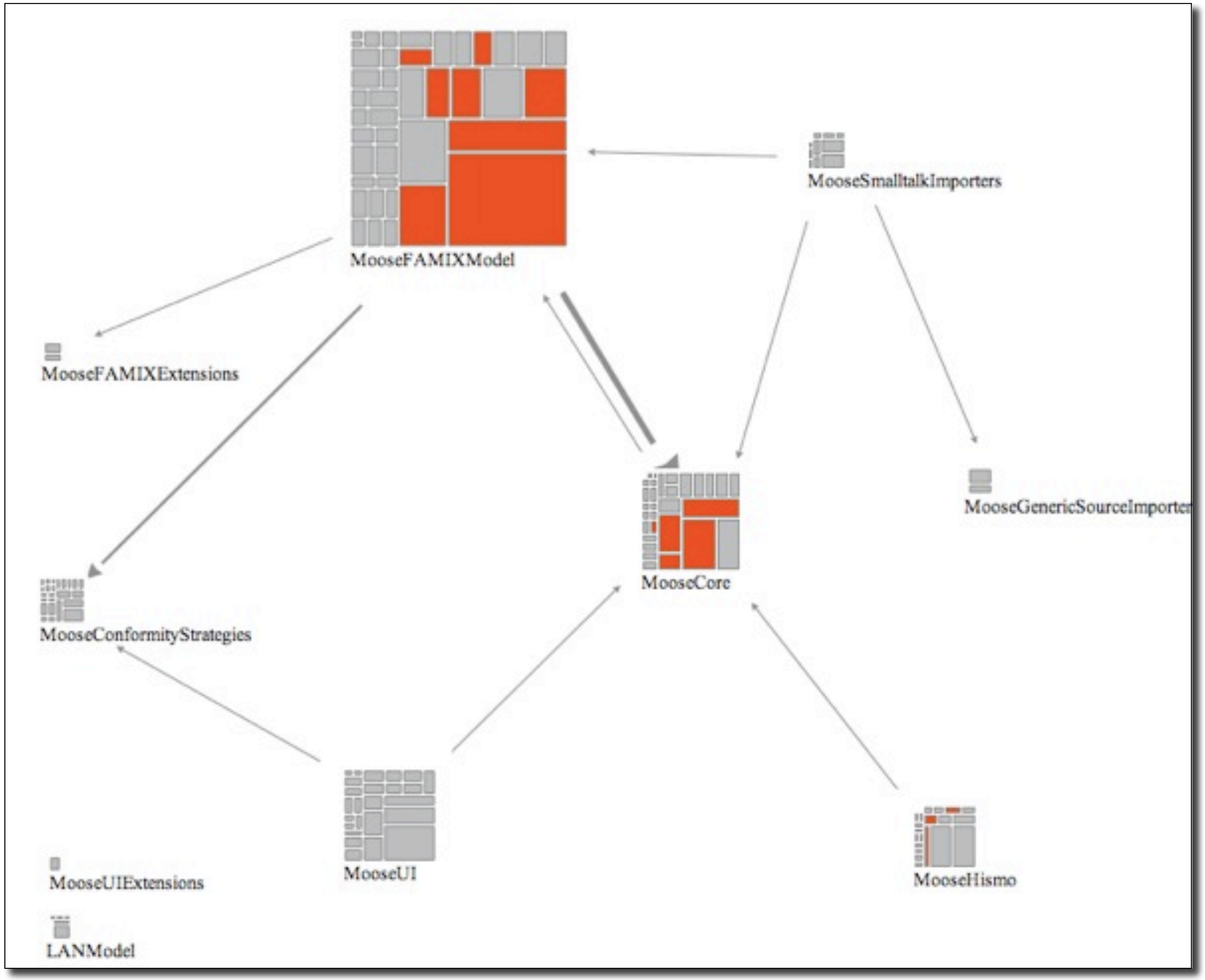
SCG 2007

# System Architecture in an Ecosystem Context



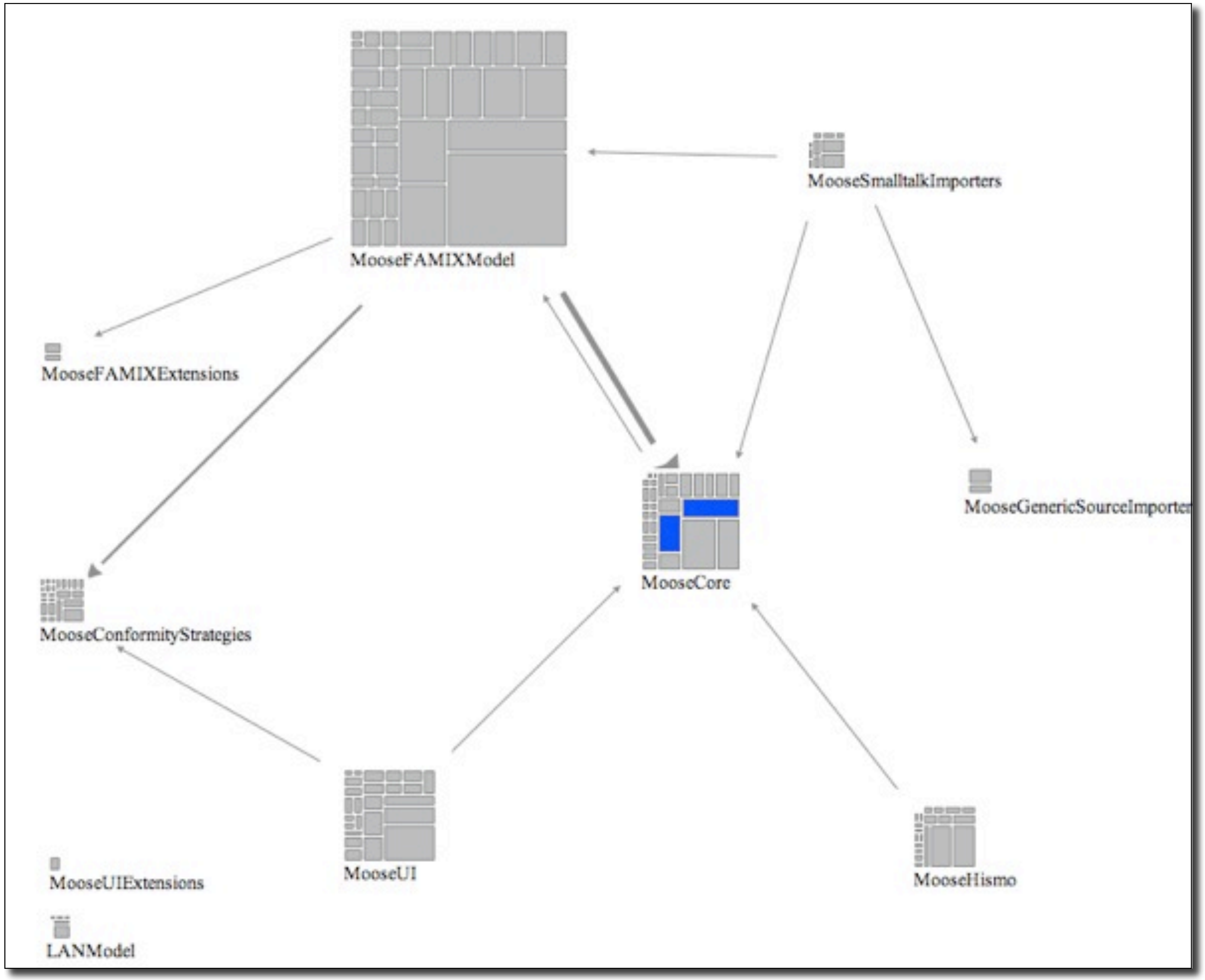
Moose 2007

# System Architecture in an Ecosystem Context



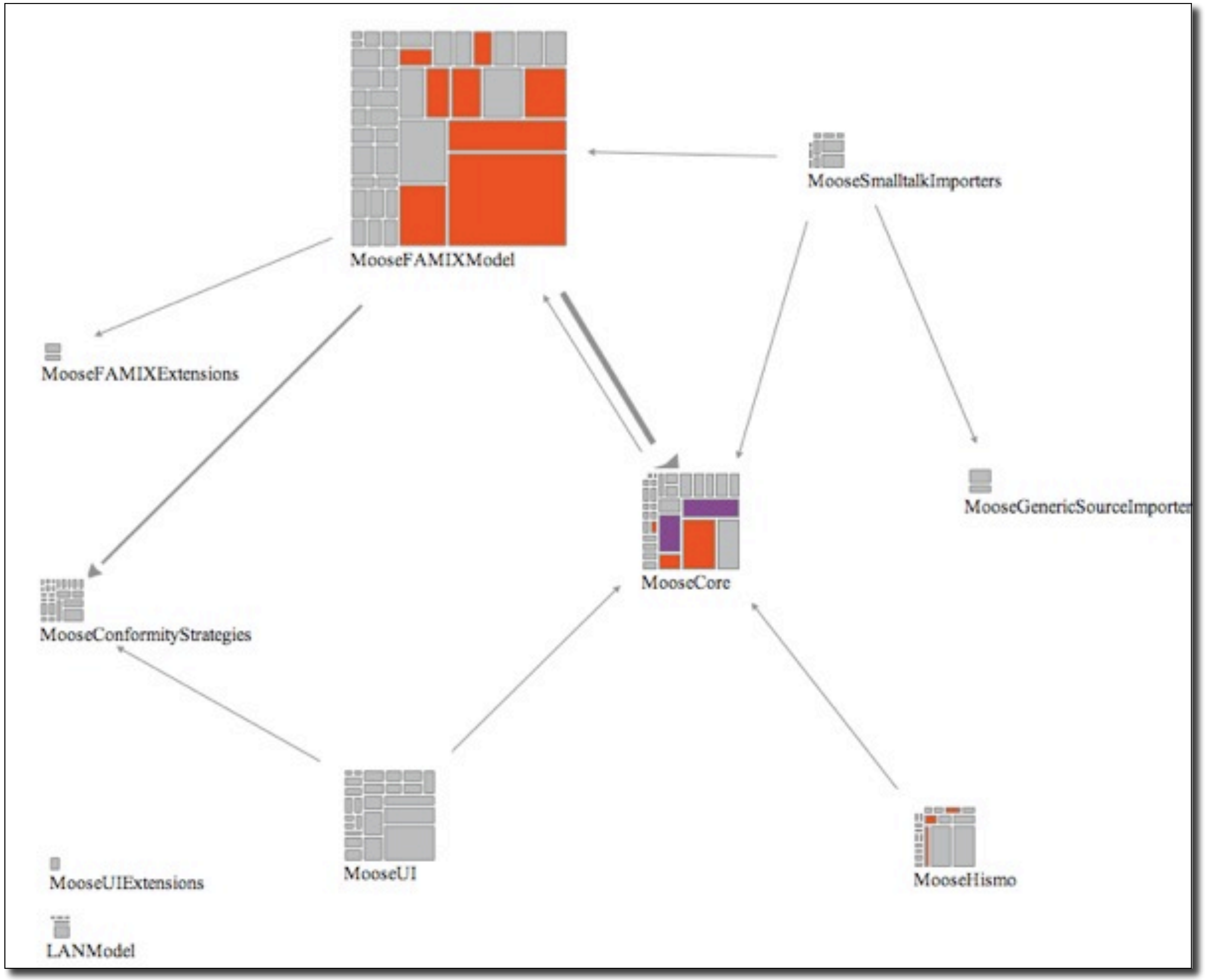
Moose 2007

# System Architecture in an Ecosystem Context



Moose 2007

# System Architecture in an Ecosystem Context



Moose 2007

# Open Questions

- How do you extract dependencies if they are not explicit?
- If they are explicit, how do you extract the details?

- Can the ecosystem provide information useful for choosing between two alternative libraries?

# Roadmap

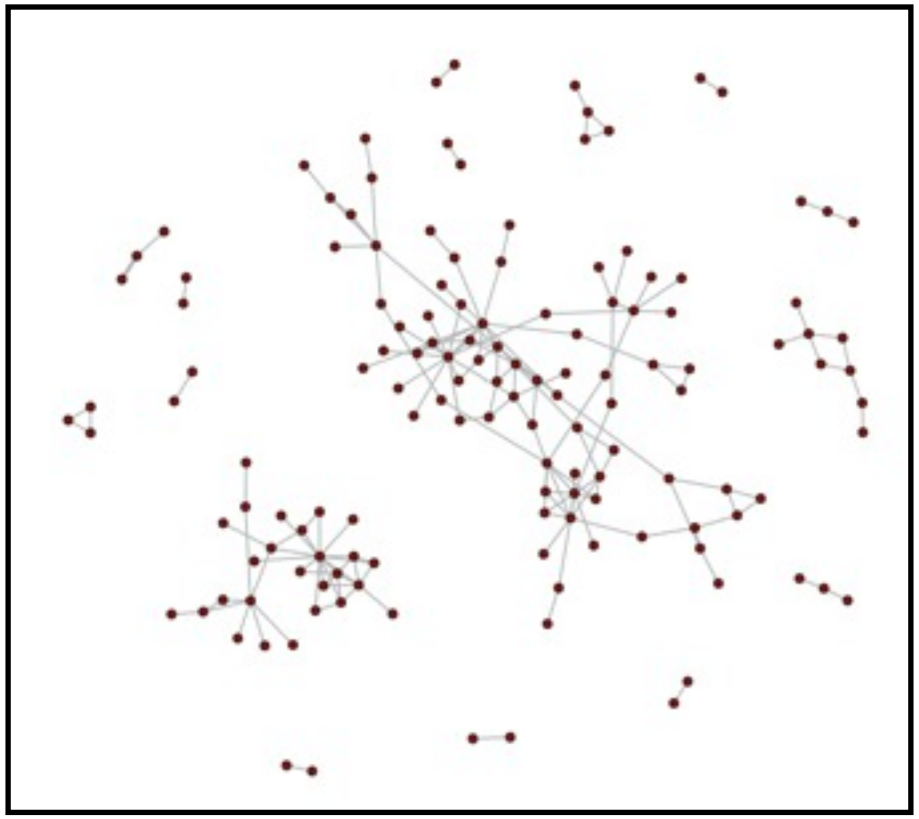
> Software Ecosystems
> Reverse Engineering Software Ecosystems
> **Dependency Analysis**
> API Evolution
> And more...

# Recovering inter-project dependencies in Software Ecosystems

> Lungu & Robbes, 2010

> Goal
  - *Evaluate techniques for automatic dependency resolution*

> Context
  - dynamic language analysis
  - unreliable declared dependencies



# The Ecco Meta-Model

- A meta-model for software ecosystems
  - lightweight
  - required entities
  - uniquely provided entities

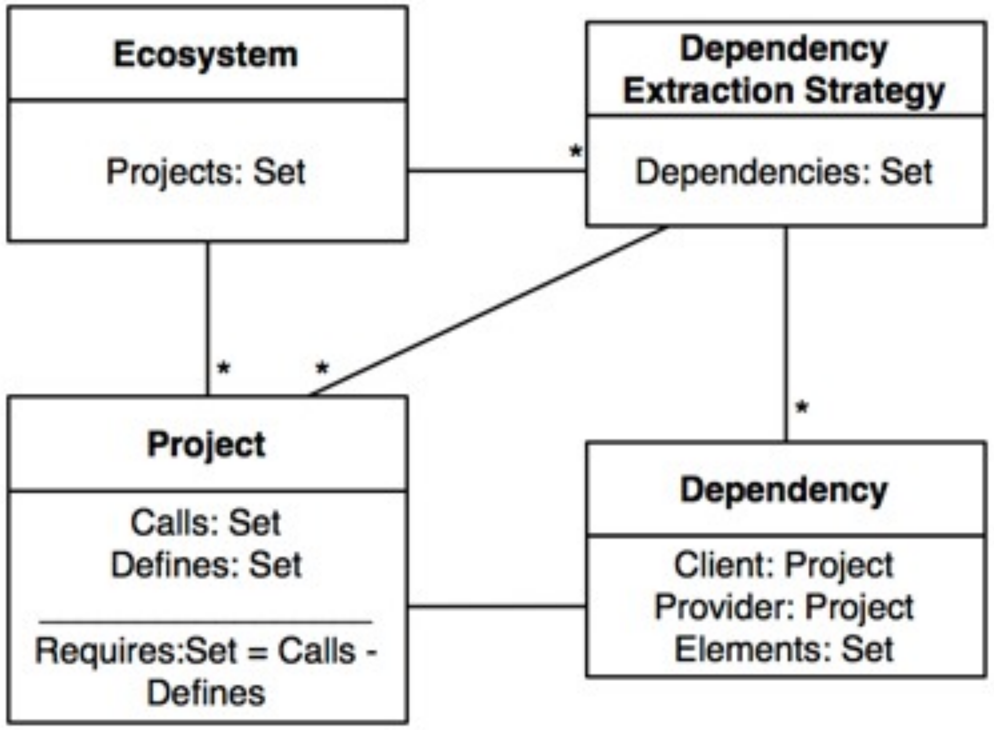


Figure 1: The *Ecco* metamodel

# Strategies for dependency detection based on Ecco

- Case Study: The Squeak 3.10 Universe
  - *Declared dependencies used as oracle*
  - *Over 200 projects*
- Approaches based on Class names are simple and performant
- False positives: 12/17 were actually true positives.
  - *You can't trust declared dependencies.*

| | Precision | Recall | F-Measure |
|---|---|---|---|
| **Unique Method Invocations** | 0.19 | 0.59 | 0.29 |
| **Unique Class References** | 0.80 | 0.71 | 0.75 |
| **Weighted Dependencies** | 0.85 | 0.70 | 0.77 |
| **Combined Method and Class References** | 0.85 | 0.70 | 0.77 |

# Strategies for dependency detection based on Ecco

> Case Study: The Squeak 3.10 Universe
  - *Declared dependencies used as oracle*
  - *Over 200 projects*

> Approaches based on Class names are simple and performant

> False positives: 12/17 were actually true positives.
  - *You can’t trust declared dependencies.*

| | Precision | Recall | F-Measure |
|---|---|---|---|
| **Unique Method Invocations** | 0.19 | 0.59 | 0.29 |
| **Unique Class References** | 0.80 | 0.71 | 0.75 |
| **Weighted Dependencies** | 0.85 | 0.70 | 0.77 |
| **Combined Method and Class References** | 0.85 | 0.70 | 0.77 |

**Lack of perfect recall is due to the incompleteness of the ecosystem**

# Automated Dependency Resolution for Open Source Software

- Ossher et al. 2010

- Goal
  - *automatically resolve the dependencies so projects can be compiled*

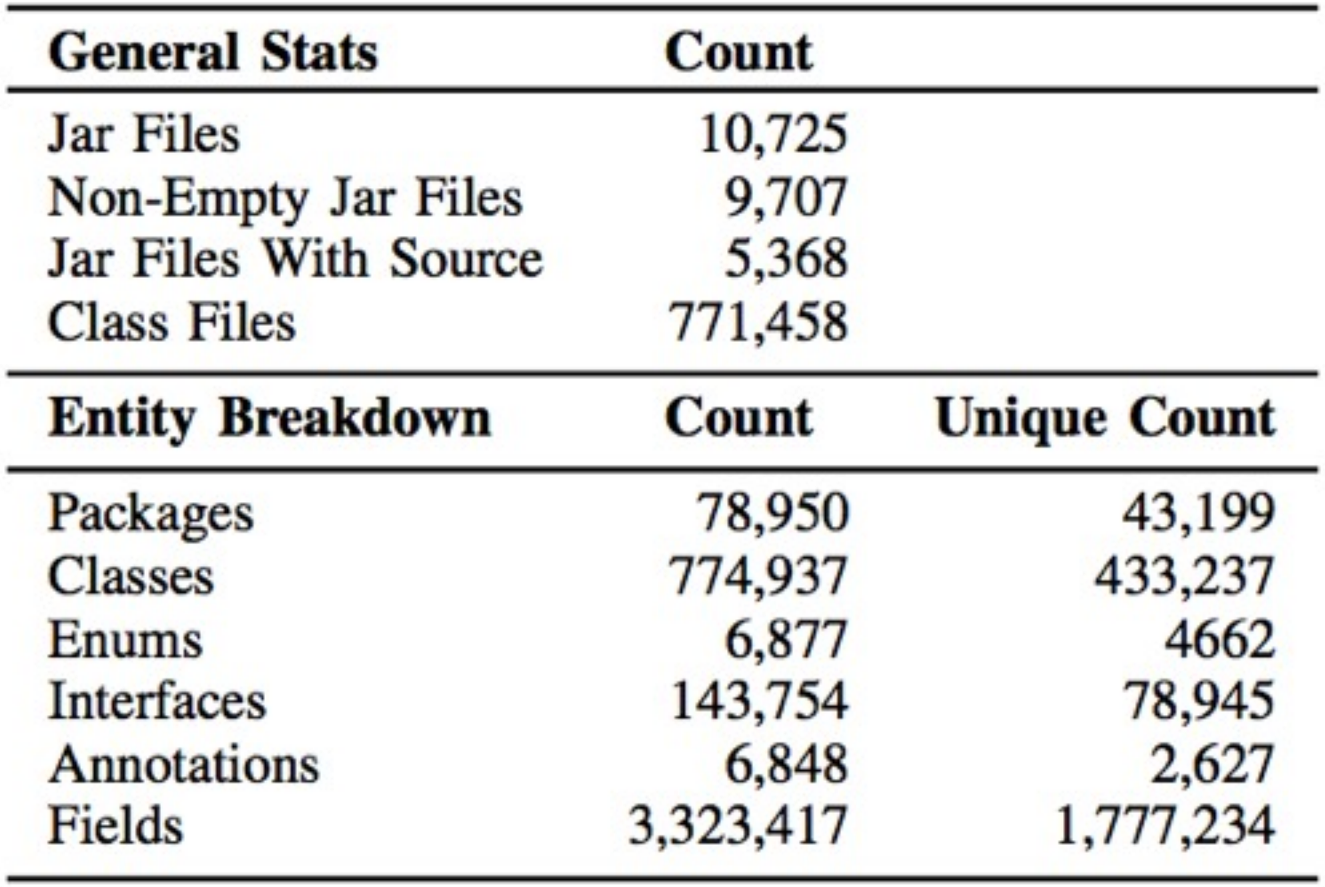
- Steps of the approach
  1. Build artifact repository
  2. Detect missing types
  3. Resolution algorithm

# Build Artifact Repository

- Case Study: Apache Maven
  - Specifies component dependencies

- Index of defined entities for every project
  - classes, interfaces
  - packages
  - enums

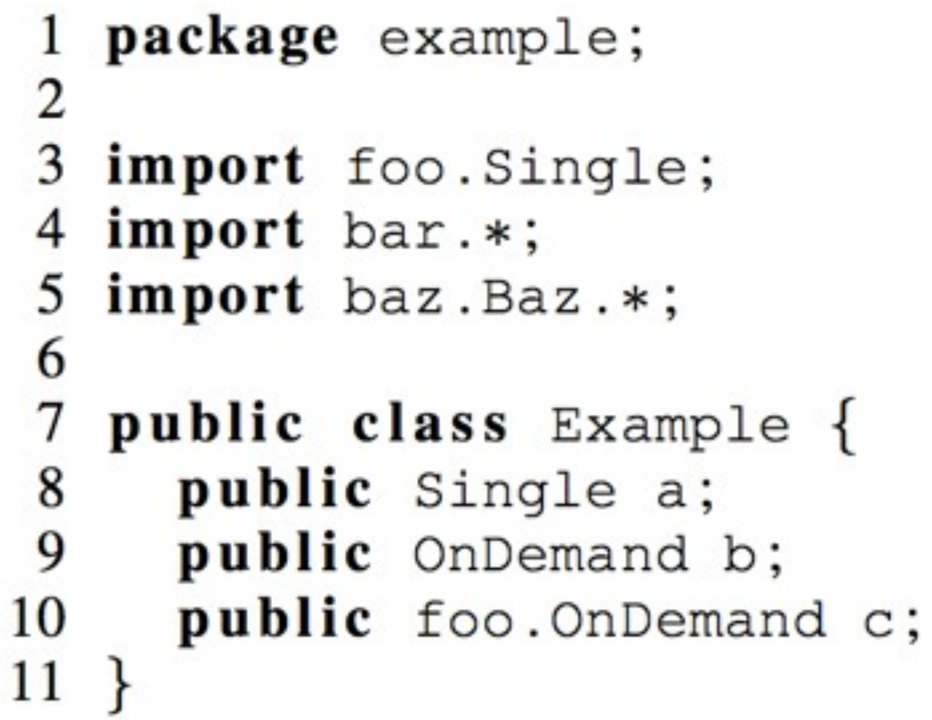
- Observation: Large amounts of duplication



| **General Stats** | **Count** | |
|---|---|---|
| Jar Files | 10,725 | |
| Non-Empty Jar Files | 9,707 | |
| Jar Files With Source | 5,368 | |
| Class Files | 771,458 | |
| **Entity Breakdown** | **Count** | **Unique Count** |
| Packages | 78,950 | 43,199 |
| Classes | 774,937 | 433,237 |
| Enums | 6,877 | 4662 |
| Interfaces | 143,754 | 78,945 |
| Annotations | 6,848 | 2,627 |
| Fields | 3,323,417 | 1,777,234 |

# Detecting Missing Types

- Restricted to
  - import statements
  - missing FQN
- FAMIX models stub entities = entities that were not found while parsing
  - *FAMIXInvocation#isStub*



```java
package example;

import foo.Single;
import bar.*;
import baz.Baz.*;

public class Example {
  public Single a;
  public OnDemand b;
  public foo.OnDemand c;
}
```

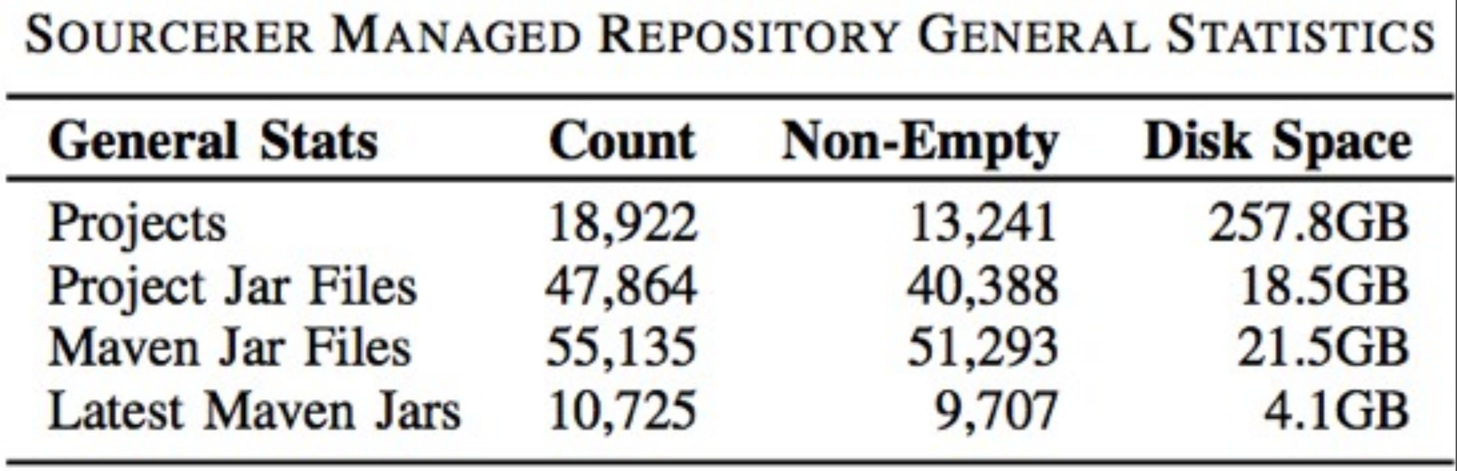
# Resolution Algorithm

- Starts with a list of FQN reported by the parser
- Uses a greedy approach

```
repeat
        - always pick the
        candidate that provides
        the most missing types
        - discount the
        artifacts provided by
        the selected candidates
until a solution is found or
there are no more candidates
```

# Case Study

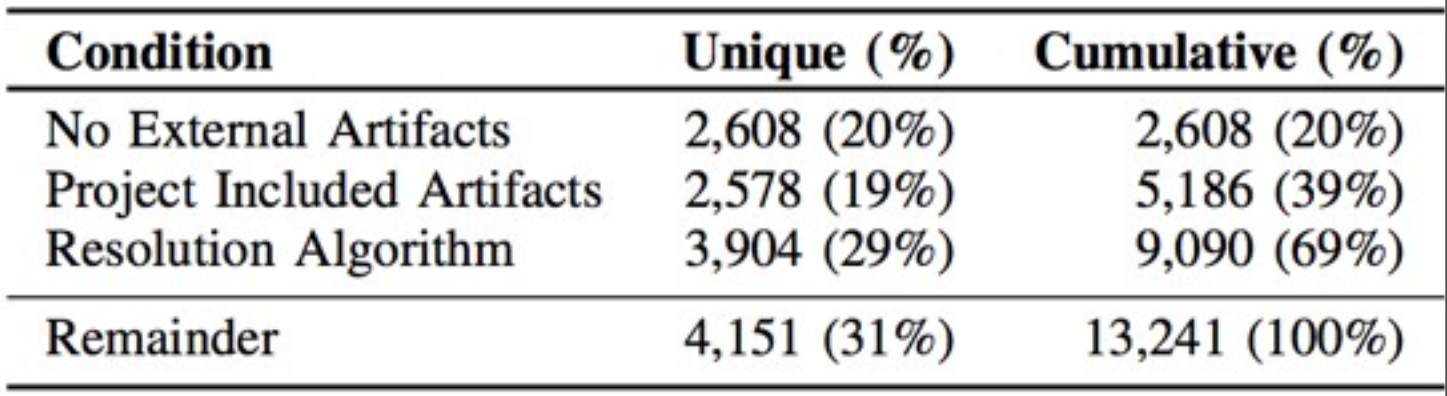
- Sourcerer DB



SOURCERER MANAGED REPOSITORY GENERAL STATISTICS

| General Stats | Count | Non-Empty | Disk Space |
|---|---|---|---|
| Projects | 18,922 | 13,241 | 257.8GB |
| Project Jar Files | 47,864 | 40,388 | 18.5GB |
| Maven Jar Files | 55,135 | 51,293 | 21.5GB |
| Latest Maven Jars | 10,725 | 9,707 | 4.1GB |

- 20% of the projects do not need external components
- 19% can be compiled with the included jar files
- **61% do not compile**



| Condition | Unique (%) | Cumulative (%) |
|---|---|---|
| No External Artifacts | 2,608 (20%) | 2,608 (20%) |
| Project Included Artifacts | 2,578 (19%) | 5,186 (39%) |
| Resolution Algorithm | 3,904 (29%) | 9,090 (69%) |
| Remainder | 4,151 (31%) | 13,241 (100%) |

# You can’t always trust declared dependencies

# You can't always trust declared dependencies

- An ecosystem can't be parsed with traditional RE tools

# You can’t always trust declared dependencies

> An ecosystem can’t be parsed with traditional RE tools

  — too large

# You can't always trust declared dependencies

> An ecosystem can't be parsed with traditional RE tools
  - too large
  - some projects don't compile

# You can't always trust declared dependencies

> An ecosystem can't be parsed with traditional RE tools
  - — too large
  - — some projects don't compile

> One needs to recover the big picture from the fragments

# You can't always trust declared dependencies

- An ecosystem can't be parsed with traditional RE tools
  - too large
  - some projects don't compile
- One needs to recover the big picture from the fragments
- The larger the index, the better the results

# You can’t always trust declared dependencies

- An ecosystem can’t be parsed with traditional RE tools
  - too large
  - some projects don’t compile
- One needs to recover the big picture from the fragments
- The larger the index, the better the results

# You can’t always trust declared dependencies

- An ecosystem can’t be parsed with traditional RE tools
  - too large
  - some projects don’t compile
- One needs to recover the big picture from the fragments
- The larger the index, the better the results

- Duplication between projects introduces noise

# You can’t always trust declared dependencies

- An ecosystem can’t be parsed with traditional RE tools
  - too large
  - some projects don’t compile
- One needs to recover the big picture from the fragments
- The larger the index, the better the results

- Duplication between projects introduces noise
  - How often does duplication happen?

# You can’t always trust declared dependencies

- An ecosystem can’t be parsed with traditional RE tools
  - too large
  - some projects don’t compile
- One needs to recover the big picture from the fragments
- The larger the index, the better the results

- Duplication between projects introduces noise
  - How often does duplication happen?
  - How to scale duplication analysis to large ecosystems?

# Roadmap

- Software Ecosystems
- Reverse Engineering Software Ecosystems
- Dependency Analysis
- **API Evolution**
- And more...

# Ripple Effects in Software Ecosystems

- Robbes & Lungu, ’11
- Problems
  - *How do you detect ripple effects?*
  - *How often do ripple effects happen?*
  - *How bad is it when they do?*
- Context
  - The study of Dig which shows that 90% of breaking changes are refactorings

# When you don’t know your clients...

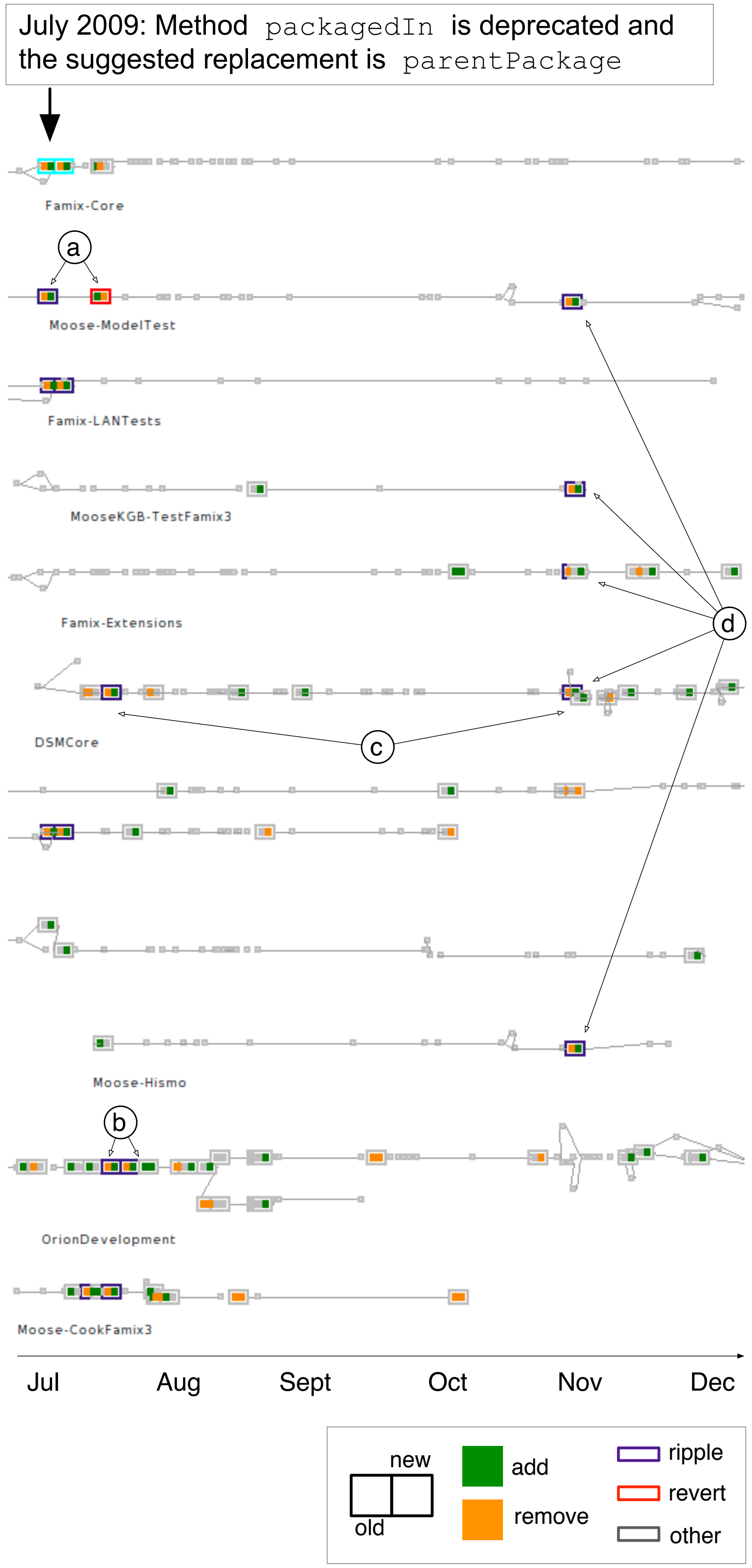
**Seaside User:** I noticed that the Seaside 2.6 dialog classes listed below are not in Seaside 2.8a1.390. [...] I am wondering if these classes have been dropped, have not been ported to 2.8 or does their functionality exists elsewhere?

**Seaside Developer:** They have been dropped. A mail went out to this list if anybody still used them and nobody replied. [...] Personally I don’t know of any application that uses these dialogs.

# Ripple Effects

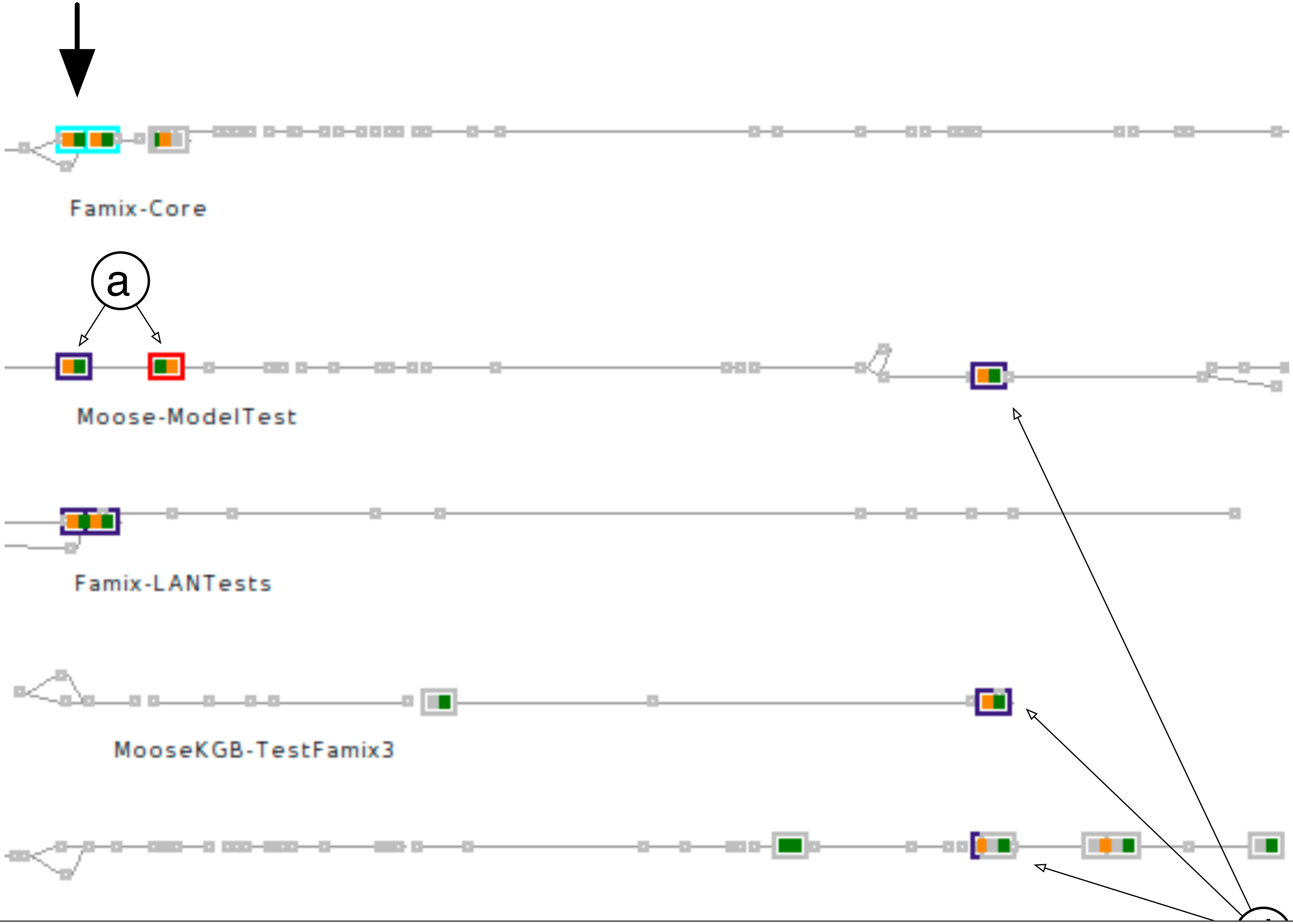
> A ripple effect is a change to a software system’s API which propagates to other systems

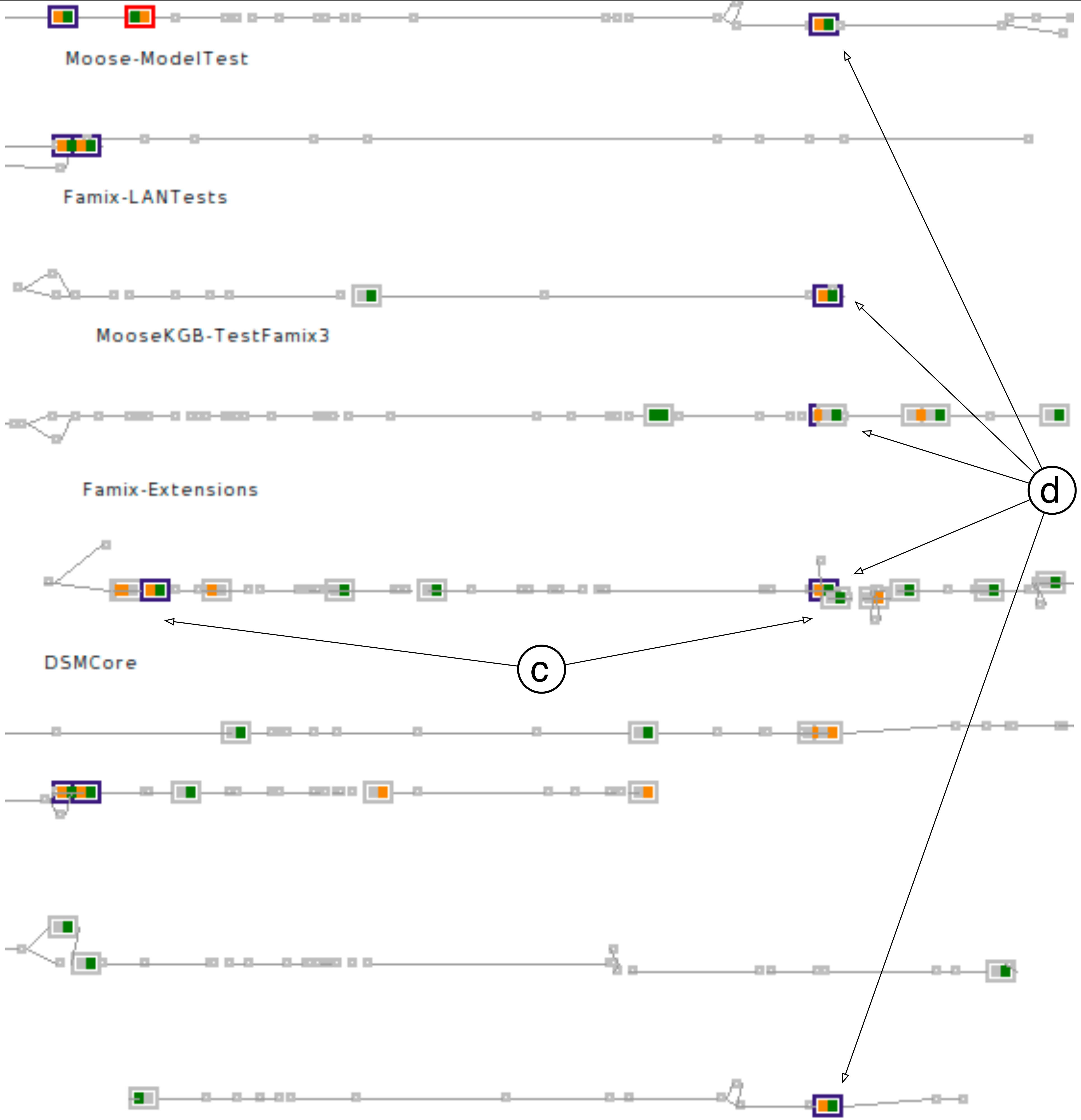
> Example: the renaming of `packagedIn`



July 2009: Method `packagedIn` is deprecated and the suggested replacement is `parentPackage`

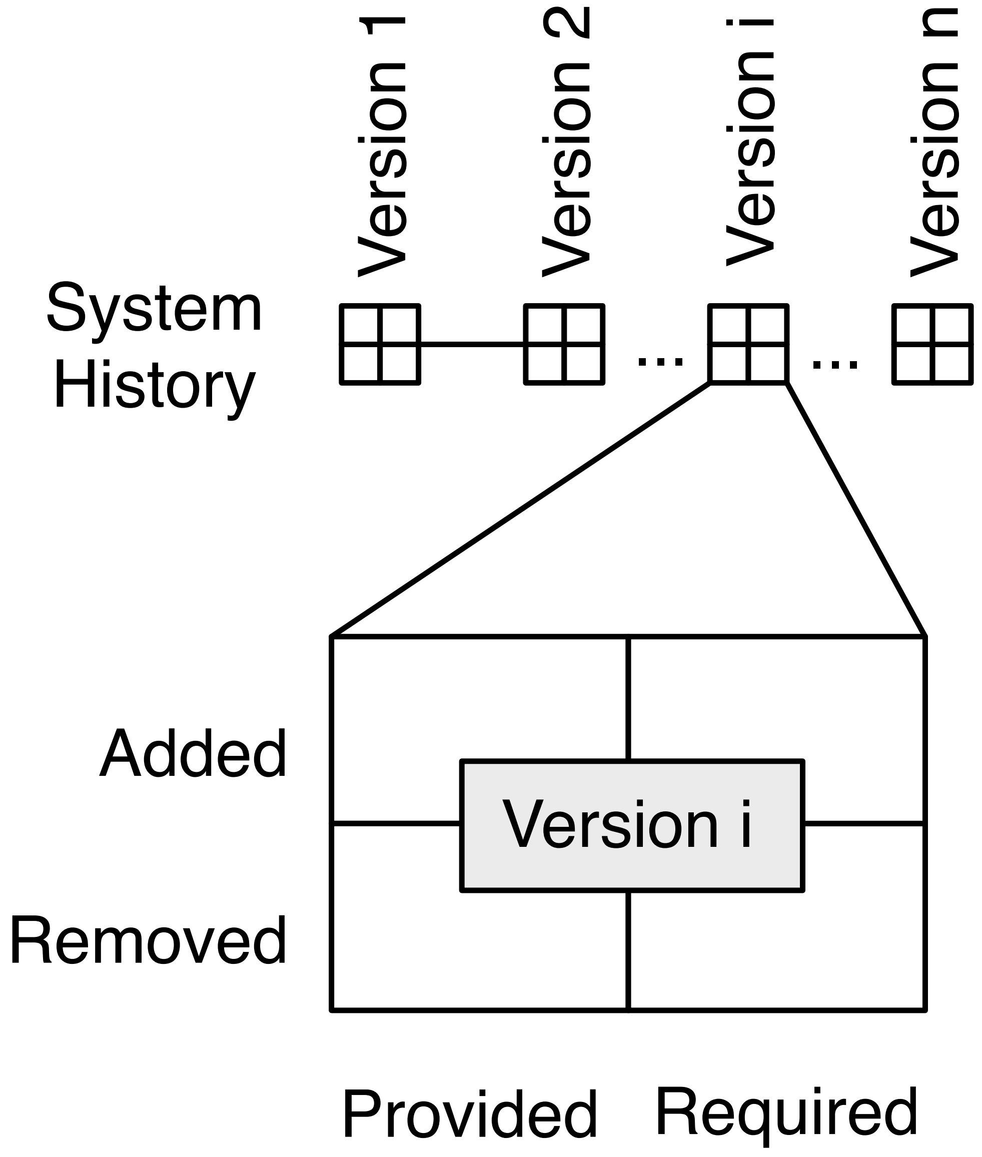
July 2009: Method `packagedIn` is deprecated and the suggested replacement is `parentPackage`




Moose-ModelTest
Famix-LANTests
MooseKGB-TestFamix3
Famix-Extensions
DSMCore
c
d

# The *Ecco-Evol* Meta-Model

- Lightweight
- Extensible
- At the project level it models only the differences between versions



# Results from Analyzing SqueakSource

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).
2. Ripples can **appear long after the original change** is introduced (more than three months).

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).
2. Ripples can **appear long after the original change** is introduced (more than three months).
3. Parts of the system can remain in **an inconsistent state for a long time** (the changes do not propagate at once in the entire dependent system).

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).
2. Ripples can **appear long after the original change** is introduced (more than three months).
3. Parts of the system can remain in **an inconsistent state for a long time** (the changes do not propagate at once in the entire dependent system).
4. **Ripples can be active over very long periods** of time in which various projects adapt to the new API.

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).
2. Ripples can **appear long after the original change** is introduced (more than three months).
3. Parts of the system can remain in **an inconsistent state for a long time** (the changes do not propagate at once in the entire dependent system).
4. **Ripples can be active over very long periods** of time in which various projects adapt to the new API.

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).
2. Ripples can **appear long after the original change** is introduced (more than three months).
3. Parts of the system can remain in **an inconsistent state for a long time** (the changes do not propagate at once in the entire dependent system).
4. **Ripples can be active over very long periods** of time in which various projects adapt to the new API.

6. **Replacements for a deprecated method can be revealed through ecosystem analysis** for replacements performed for that method.

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).
2. Ripples can **appear long after the original change** is introduced (more than three months).
3. Parts of the system can remain in **an inconsistent state for a long time** (the changes do not propagate at once in the entire dependent system).
4. **Ripples can be active over very long periods** of time in which various projects adapt to the new API.

6. **Replacements for a deprecated method can be revealed through ecosystem analysis** for replacements performed for that method.
7. Often **systems remain dependent on deprecated methods**. Some are dead and some remain dependent on older versions of the required system.

# Results from Analyzing SqueakSource

1. Ripples can have a **very large impact** on the ecosystem (in terms of projects or developers that are impacted by the change).
2. Ripples can **appear long after the original change** is introduced (more than three months).
3. Parts of the system can remain in **an inconsistent state for a long time** (the changes do not propagate at once in the entire dependent system).
4. **Ripples can be active over very long periods** of time in which various projects adapt to the new API.

6. **Replacements for a deprecated method can be revealed through ecosystem analysis** for replacements performed for that method.
7. Often **systems remain dependent on deprecated methods**. Some are dead and some remain dependent on older versions of the required system.
8. **Developers defensively deprecate a large number of methods** that are never used outside their project.

# Mining Framework Usage Changes from Instantiation Code

> Schaeffer et al., ’08

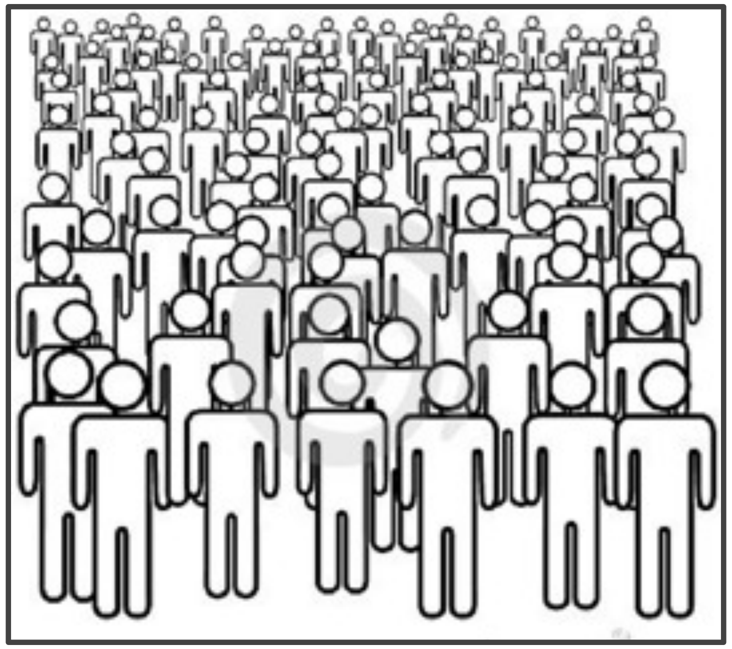
> Goal

- *Suggest changes to support evolution based on the changes of the early adopters*

> Context

— Ripple effects break the clients

— Other approaches look at the evolution of the framework itself [Dagenais & Robillard]



# Overview

- Framework Instantiations = other systems that use the framework

- Extract rules that show how to adapt to framework evolution

- Steps
  1. Fact Extraction
  2. Creating Transactions
  3. Extracting Rules

Example rule:

**Calls to method Plugin.shutdown() are replaced to calls to method Plugin.stop()**

# Fact Extraction

- Facts are groupFacts for a given class “T”
  - — Extends: FT
  - — Implements: FT
  - — Overrides: FT.m()
  - — Instantiates: FT
  - — Calls: FT.m()
  - — Accesses: FT.

- T inherits facts from superclasses

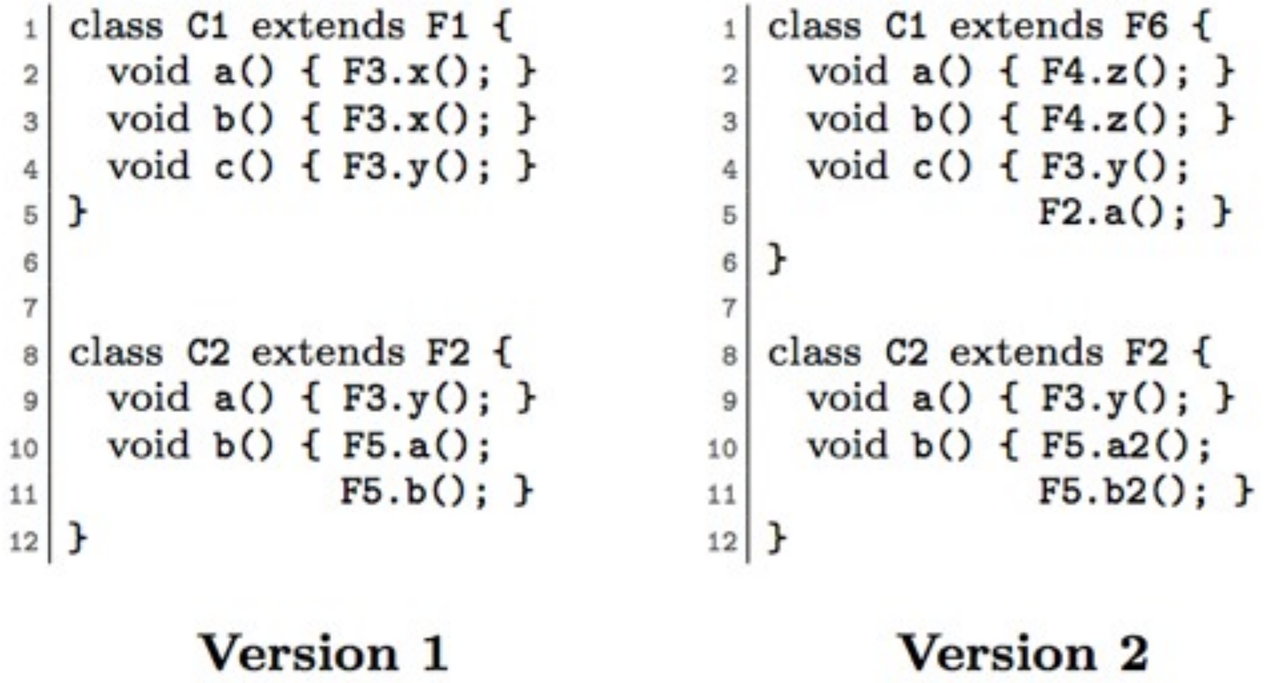
# Creating Transactions

> Straightforward approach:

— one transaction per instantiation class

> Actual approach:

1. partitioning the usage based on **contexts**
   - *class declaration*
   - *each method*
   - *allows a more focused analysis*
     - facts extracted from m1 in v1 are not relevant to facts in n2 in v2
     - example: c1.a() -> F4.z()

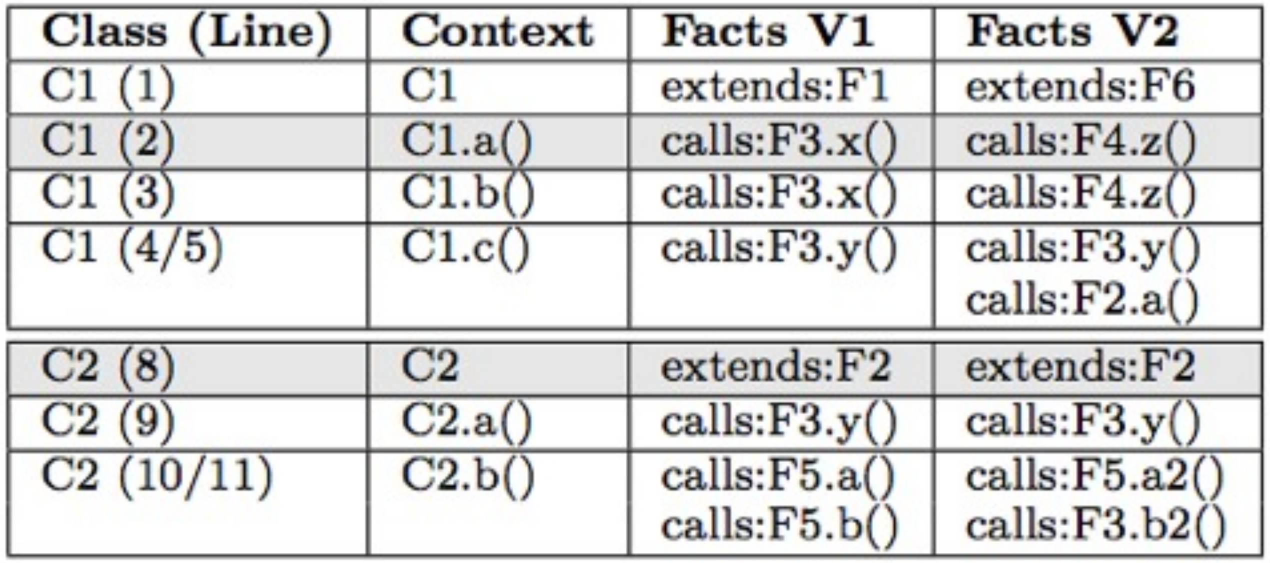


```
 1 class C1 extends F1 {
 2   void a() { F3.x(); }
 3   void b() { F3.x(); }
 4   void c() { F3.y(); }
 5 }
 6
 7
 8 class C2 extends F2 {
 9   void a() { F3.y(); }
10   void b() { F5.a();
11              F5.b(); }
12 }
```

**Version 1**

```
 1 class C1 extends F6 {
 2   void a() { F4.z(); }
 3   void b() { F4.z(); }
 4   void c() { F3.y();
 5              F2.a(); }
 6 }
 7
 8 class C2 extends F2 {
 9   void a() { F3.y(); }
10   void b() { F5.a2();
11              F5.b2(); }
12 }
```

**Version 2**

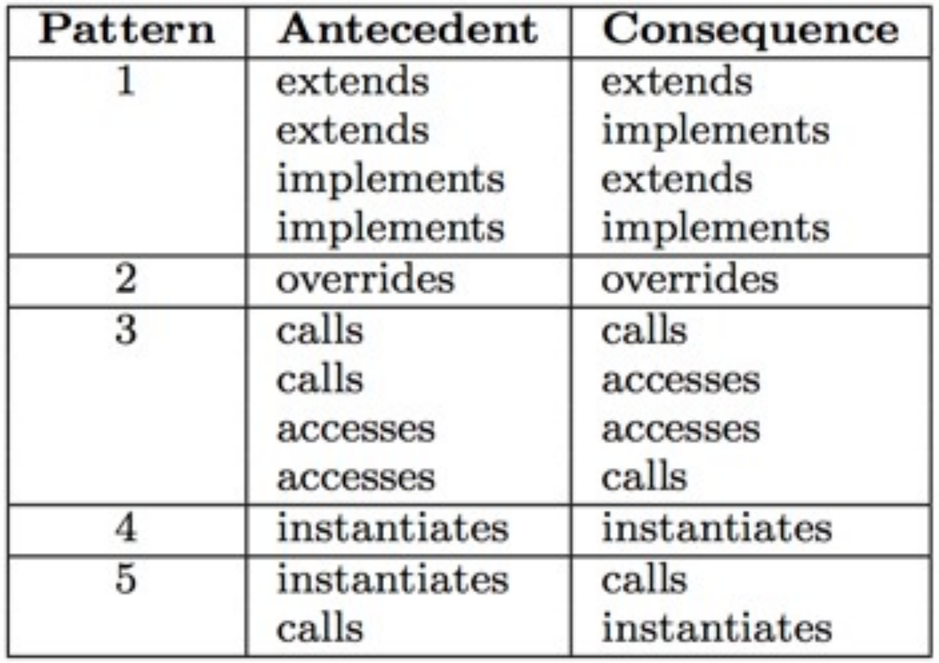


| Class (Line) | Context | Facts V1 | Facts V2 |
|---|---|---|---|
| C1 (1) | C1 | extends:F1 | extends:F6 |
| C1 (2) | C1.a() | calls:F3.x() | calls:F4.z() |
| C1 (3) | C1.b() | calls:F3.x() | calls:F4.z() |
| C1 (4/5) | C1.c() | calls:F3.y() | calls:F3.y()<br>calls:F2.a() |
| C2 (8) | C2 | extends:F2 | extends:F2 |
| C2 (9) | C2.a() | calls:F3.y() | calls:F3.y() |
| C2 (10/11) | C2.b() | calls:F5.a()<br>calls:F5.b() | calls:F5.a2()<br>calls:F3.b2() |

**Table 1: Extracted facts**

# Actual Approach (cont’d)

2. Taking change patterns into consideration
3. Removing unchanged usages



| Pattern | Antecedent | Consequence |
|---|---|---|
| 1 | extends | extends |
| | extends | implements |
| | implements | extends |
| | implements | implements |
| 2 | overrides | overrides |
| 3 | calls | calls |
| | calls | accesses |
| | accesses | accesses |
| | accesses | calls |
| 4 | instantiates | instantiates |
| 5 | instantiates | calls |
| | calls | instantiates |

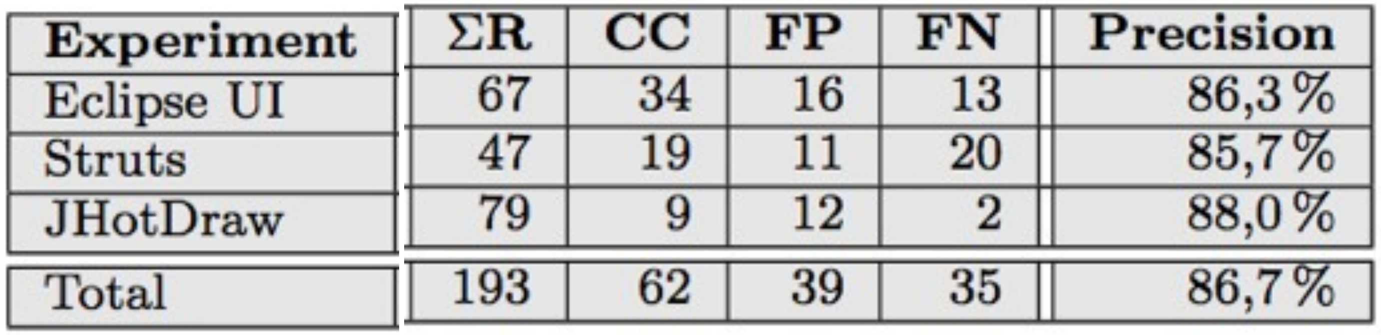
**Table 2: Five categories of change patterns**

# Pattern Extraction

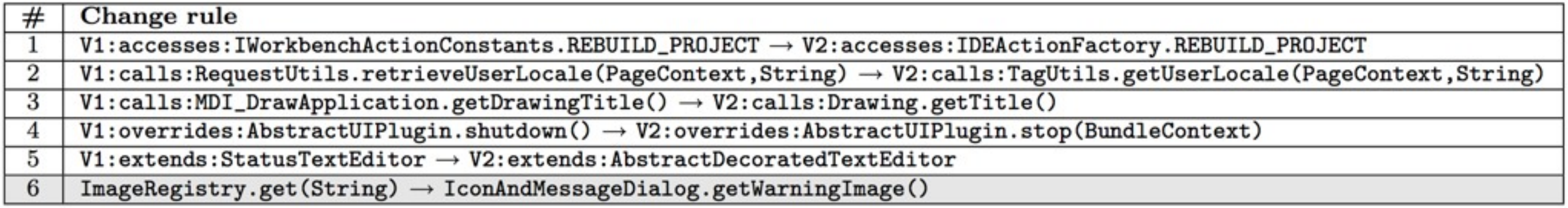
- Minimum confidence
  - how often the two items appear together
- Minimum support
  - how often if the *antecedent* is in also the *consequence* is in
- Consider only patterns that have one antecedent and one consequence

# Evaluation

- 3/4 changes caused by refactorings
- 1/4 changes not caused by refactorings
- 39 false positives



| Experiment | ΣR | CC | FP | FN | Precision |
|---|---|---|---|---|---|
| Eclipse UI | 67 | 34 | 16 | 13 | 86,3 % |
| Struts | 47 | 19 | 11 | 20 | 85,7 % |
| JHotDraw | 79 | 9 | 12 | 2 | 88,0 % |
| Total | 193 | 62 | 39 | 35 | 86,7 % |



| # | Change rule |
|---|---|
| 1 | V1:accesses:IWorkbenchActionConstants.REBUILD_PROJECT → V2:accesses:IDEActionFactory.REBUILD_PROJECT |
| 2 | V1:calls:RequestUtils.retrieveUserLocale(PageContext,String) → V2:calls:TagUtils.getUserLocale(PageContext,String) |
| 3 | V1:calls:MDI_DrawApplication.getDrawingTitle() → V2:calls:Drawing.getTitle() |
| 4 | V1:overrides:AbstractUIPlugin.shutdown() → V2:overrides:AbstractUIPlugin.stop(BundleContext) |
| 5 | V1:extends:StatusTextEditor → V2:extends:AbstractDecoratedTextEditor |
| 6 | ImageRegistry.get(String) → IconAndMessageDialog.getWarningImage() |

# Discussion

- Assumptions
  - Users of the framework that have adapted should already exist
  - Transactions can be built for program elements that exist in both the versions
  - Usage changes are limited to one antecedent one consequence
- Threats to validity
  - External validity = do the results generalize?
  - Internal validity = is the analysis correct?
    - *e.g. evaluator bias*

# Roadmap

- Software Ecosystems
- Reverse Engineering Software Ecosystems
- Dependency Analysis
- API Evolution
- **And more...**

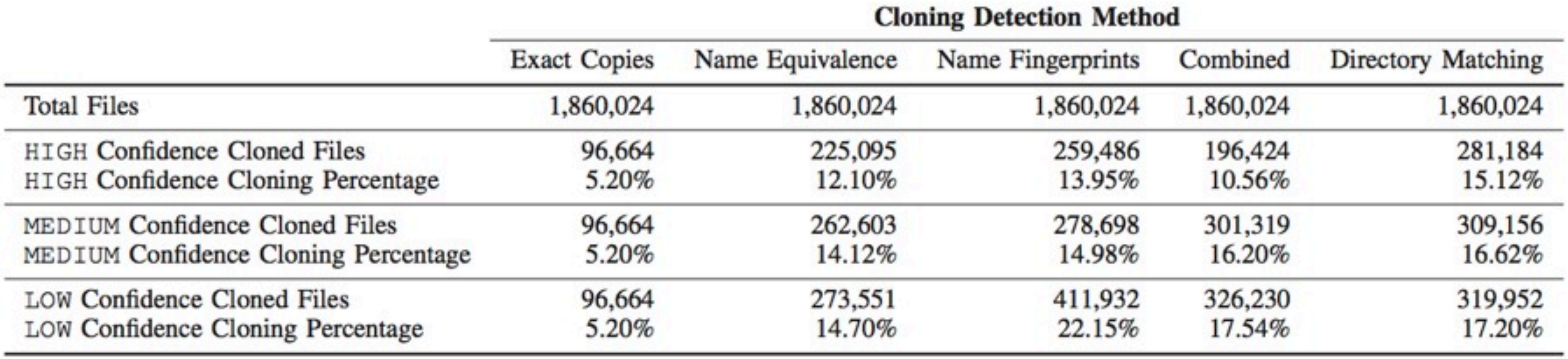
# Clone Detection

- Problems
  - Licensing information
  - Origin analysis

- Types of clones
  - Type 1: identical code fragments with the exception of whitespace and comments
  - Type 2: syntactically identical fragments except for variations in identifiers, literals, whitespace, and comments
  - Type 3: copied fragments with further modifications such as changed, added, or removed statements, in addition to variations in identifiers, literals, whitespace, and comments.

# Clone Detection

- Ossher et al. 2011
  - Analyze large corpus of Java systems from Sourcerer DB
  - Evaluate different techniques for detecting clones
    - *Exact copies: computing the hash*
    - *Name equivalence: comparing FQNs*
    - *Name fingerprints: comparing names of the structural entities inside a class*
    - *Combined: combining the previous approaches*

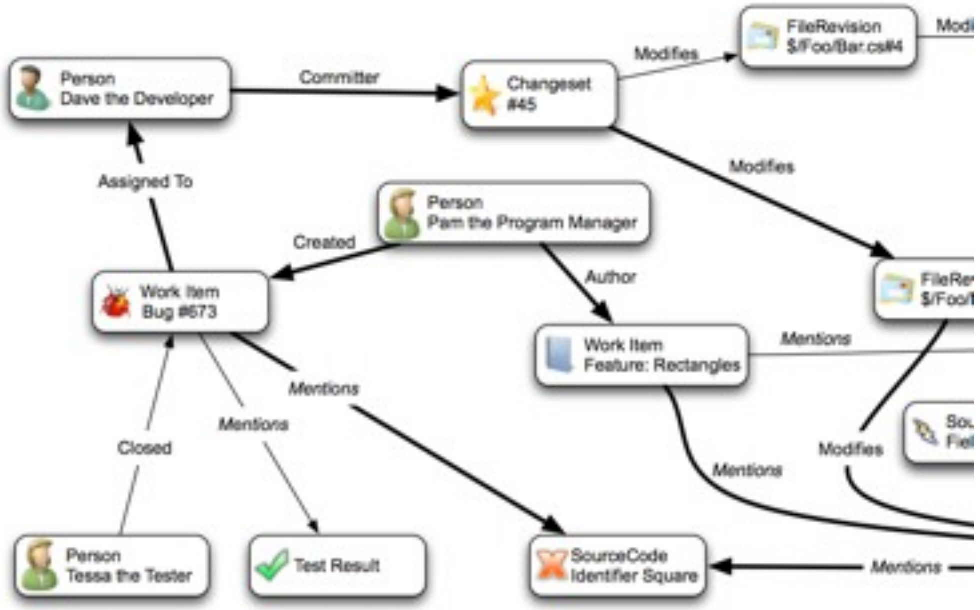


| | **Cloning Detection Method** | | | | |
|---|---|---|---|---|---|
| | Exact Copies | Name Equivalence | Name Fingerprints | Combined | Directory Matching |
| Total Files | 1,860,024 | 1,860,024 | 1,860,024 | 1,860,024 | 1,860,024 |
| HIGH Confidence Cloned Files | 96,664 | 225,095 | 259,486 | 196,424 | 281,184 |
| HIGH Confidence Cloning Percentage | 5.20% | 12.10% | 13.95% | 10.56% | 15.12% |
| MEDIUM Confidence Cloned Files | 96,664 | 262,603 | 278,698 | 301,319 | 309,156 |
| MEDIUM Confidence Cloning Percentage | 5.20% | 14.12% | 14.98% | 16.20% | 16.62% |
| LOW Confidence Cloned Files | 96,664 | 273,551 | 411,932 | 326,230 | 319,952 |
| LOW Confidence Cloning Percentage | 5.20% | 14.70% | 22.15% | 17.54% | 17.20% |

TABLE IV
FILE CLONING RATES FOR EACH DETECTION METHOD

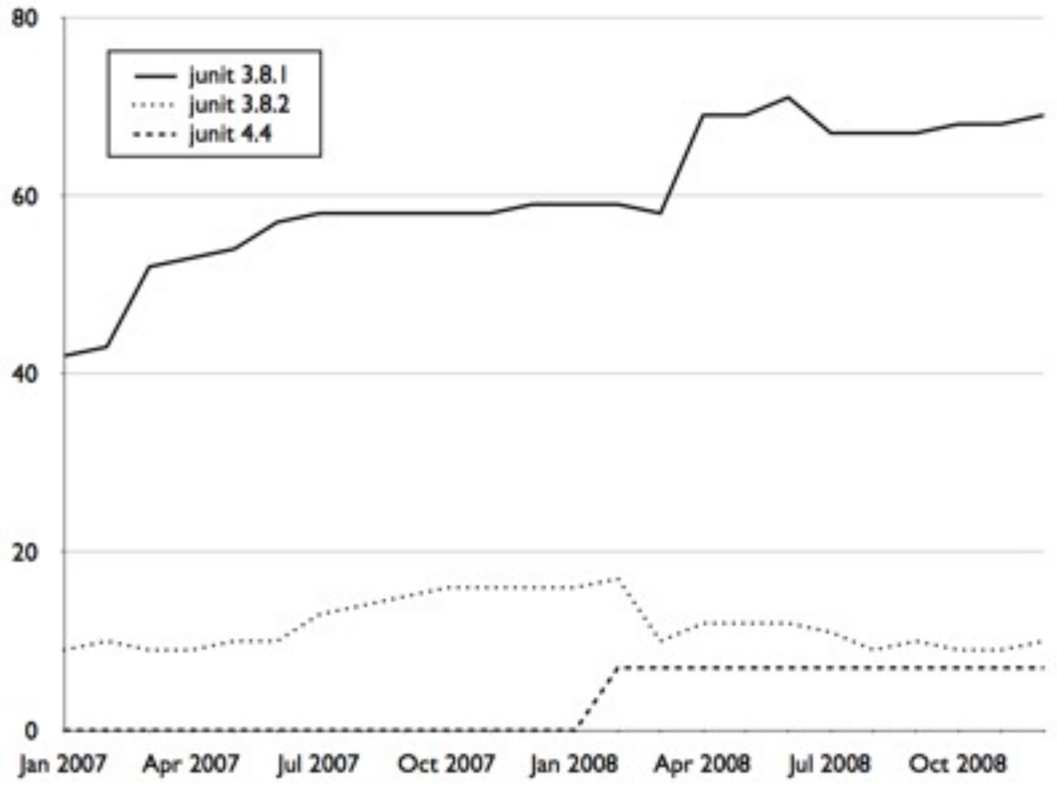
# Developer needs in the ecosystem

- Begel et al. ’10
- Survey information needs in Microsoft
    1. *Find the relevant engineers for a feature*
    2. *Find an expert on a given feature*
    3. *Find all the resources related to a given feature, API, product*
    4. *Find why a recent change was made*
    5. *Being notified that a recent change affects an engineer’s work*
    6. *Finding who might be affected by a given change to code/API*
- Codebook - social network



# Mining Trends in Library Usage

- Mileva et al., 2009
- Assumption: Popularity of libraries might be a good indicator of their quality
- Case Study
  - Apache Ecosystem (250 projects)
  - Maven-based dependencies



# Mining Trends in Library Usage (contd.)

- Assumption: Switching back from a library version might be a good indicator of quality

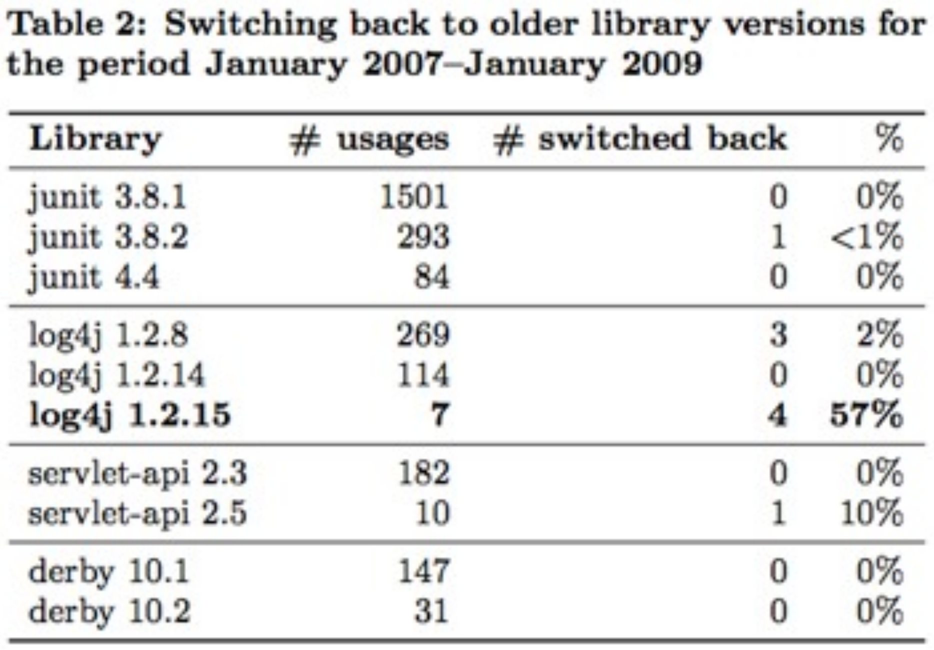
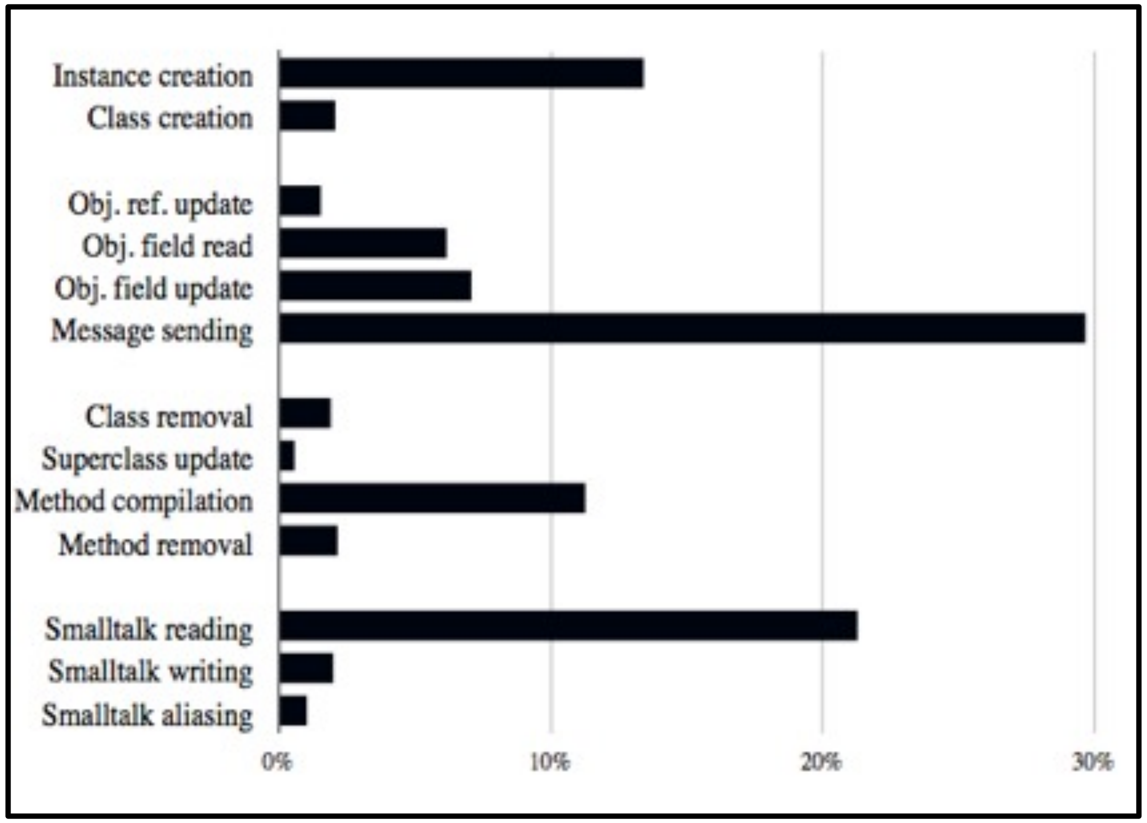


**Table 2: Switching back to older library versions for the period January 2007–January 2009**

| Library | # usages | # switched back | % |
|---|---|---|---|
| junit 3.8.1 | 1501 | 0 | 0% |
| junit 3.8.2 | 293 | 1 | <1% |
| junit 4.4 | 84 | 0 | 0% |
| log4j 1.2.8 | 269 | 3 | 2% |
| log4j 1.2.14 | 114 | 0 | 0% |
| **log4j 1.2.15** | **7** | **4** | **57%** |
| servlet-api 2.3 | 182 | 0 | 0% |
| servlet-api 2.5 | 10 | 1 | 10% |
| derby 10.1 | 147 | 0 | 0% |
| derby 10.2 | 31 | 0 | 0% |

# Learning how programmers use language features

- Callau et al. ’11
- Study the usage of reflection in SqueakSource
  - safe vs. unsafe usages
  - dynamic features are not used often
  - dynamic features are used in specific kinds of projects



# What you should know!

- What is an ecosystem
- What is the relationship between an ecosystem and a super-repository
- What are ripple effects
- What are some of the problems associated with analyzing a software ecosystem
- The Ecco-Evol meta-model

# Can you answer these questions?

> Discuss an approach for detecting inter-project dependencies in a software ecosystem. What are some of the problems and limitations?

> How can the information in an ecosystem support a client's migration from one version of a library to another one?

> What's the difference between the *Ecco-Evol* and FAMIX?

> Can you describe an approach for mining library usage from the ecosystem?

> What would be your approach to detecting clones in a large software ecosystem?

# Further Reading

- **Recommending Adaptive Changes for Framework Evolution**, Dagenais & Robillard, 2008
- **Reverse Engineering Software Ecosystems**, Lungu, 2009
- **Codebook: Discovering and Exploiting Relationships in Software Repositories,** Begel et al. 2010
- **How Developers Use the Dynamic Features of Programming Languages**, Calau et al. 2011
- **Mining Trends in Library Usage**, Mileva et al. 2009

- http://scg.unibe.ch/scgbib?query=sde-ecosystems