UNIVERSITÄT BERN

Software Ecosystems

Mircea Lungu

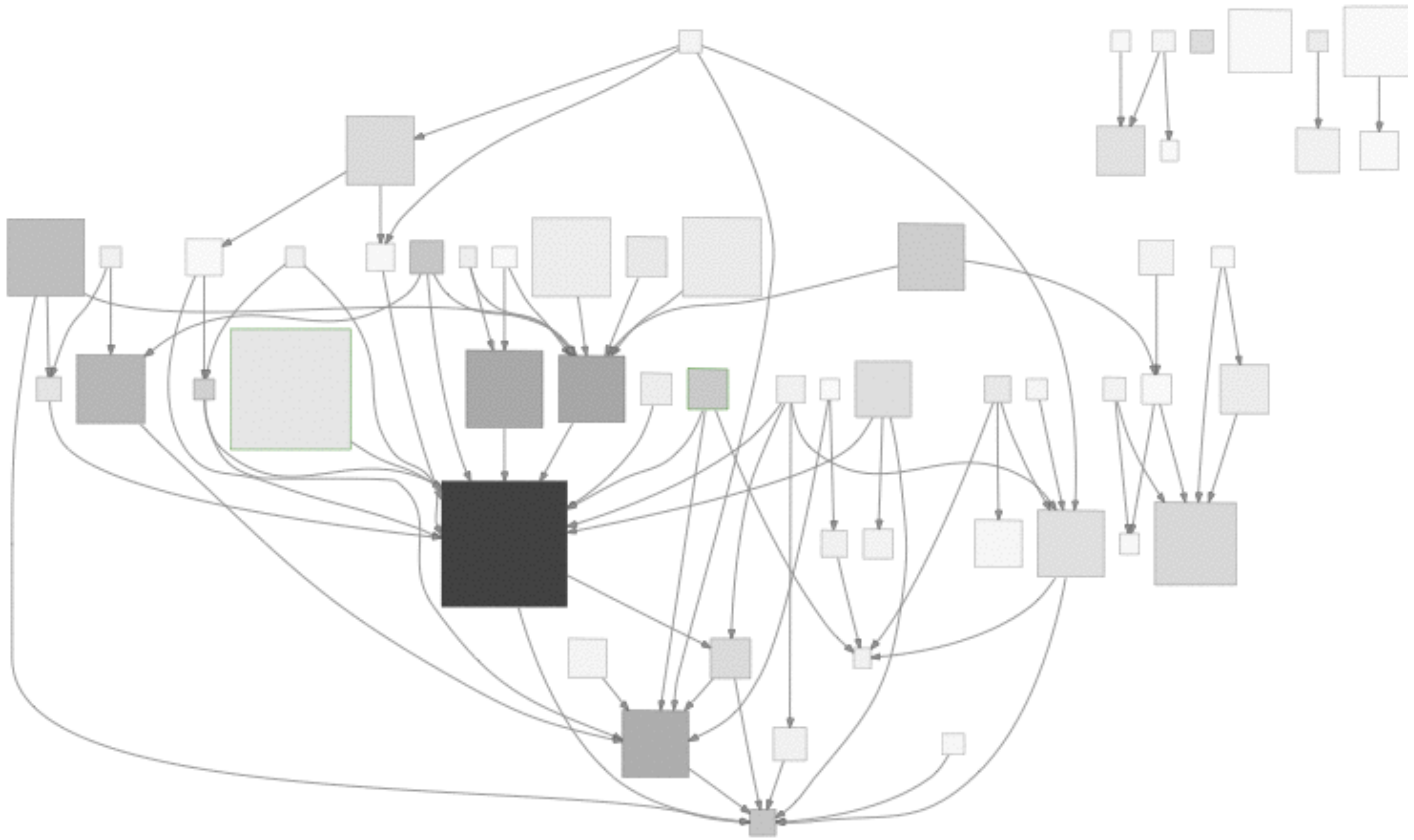
A Thousand Years in Bali

Stephen Lansing

Ecosystems

A software ecosystem is*

a set of inter-connected, independently developed, co-evolving software systems.

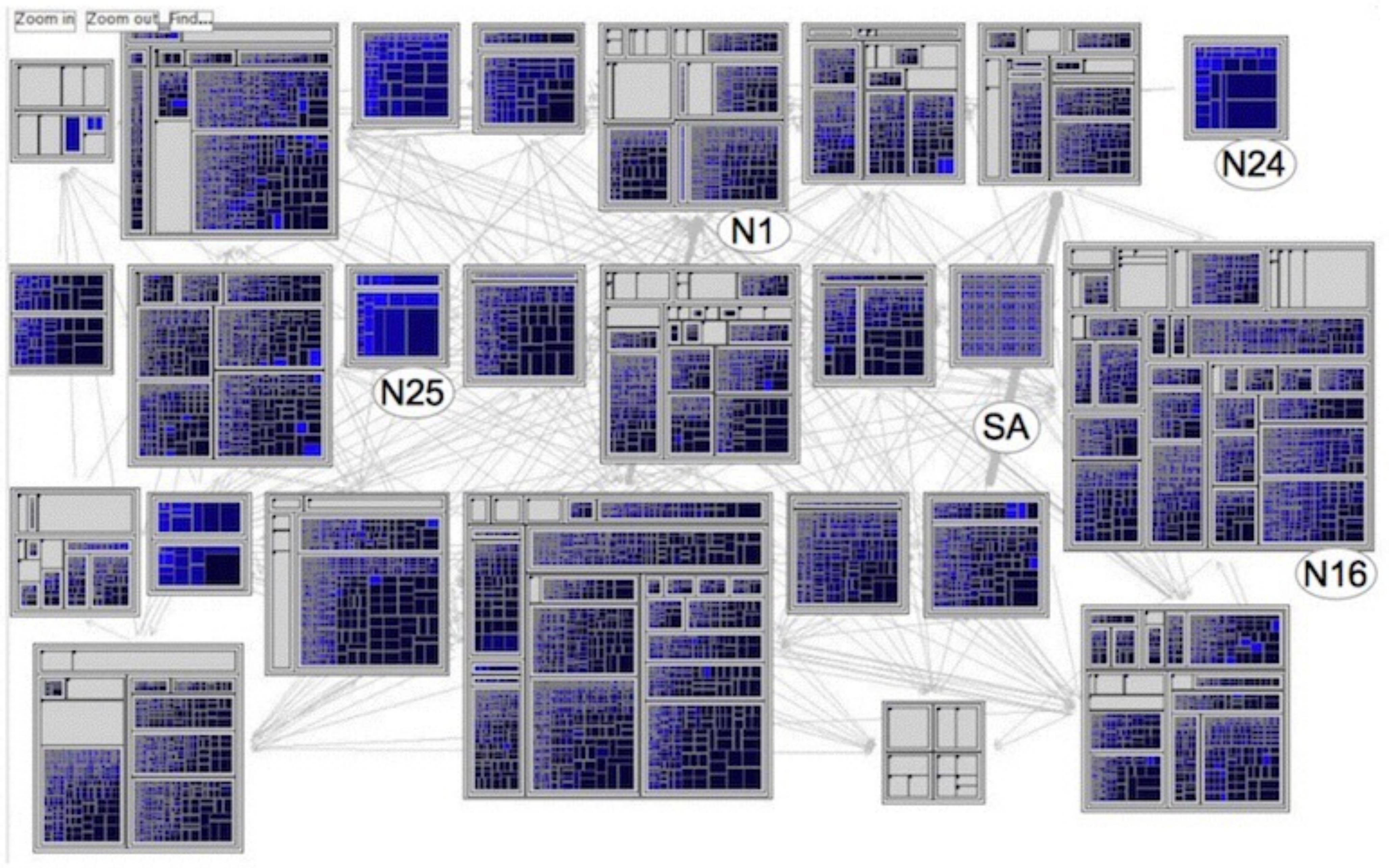


Generators of ecosystems

The PyPI Dependency Graph

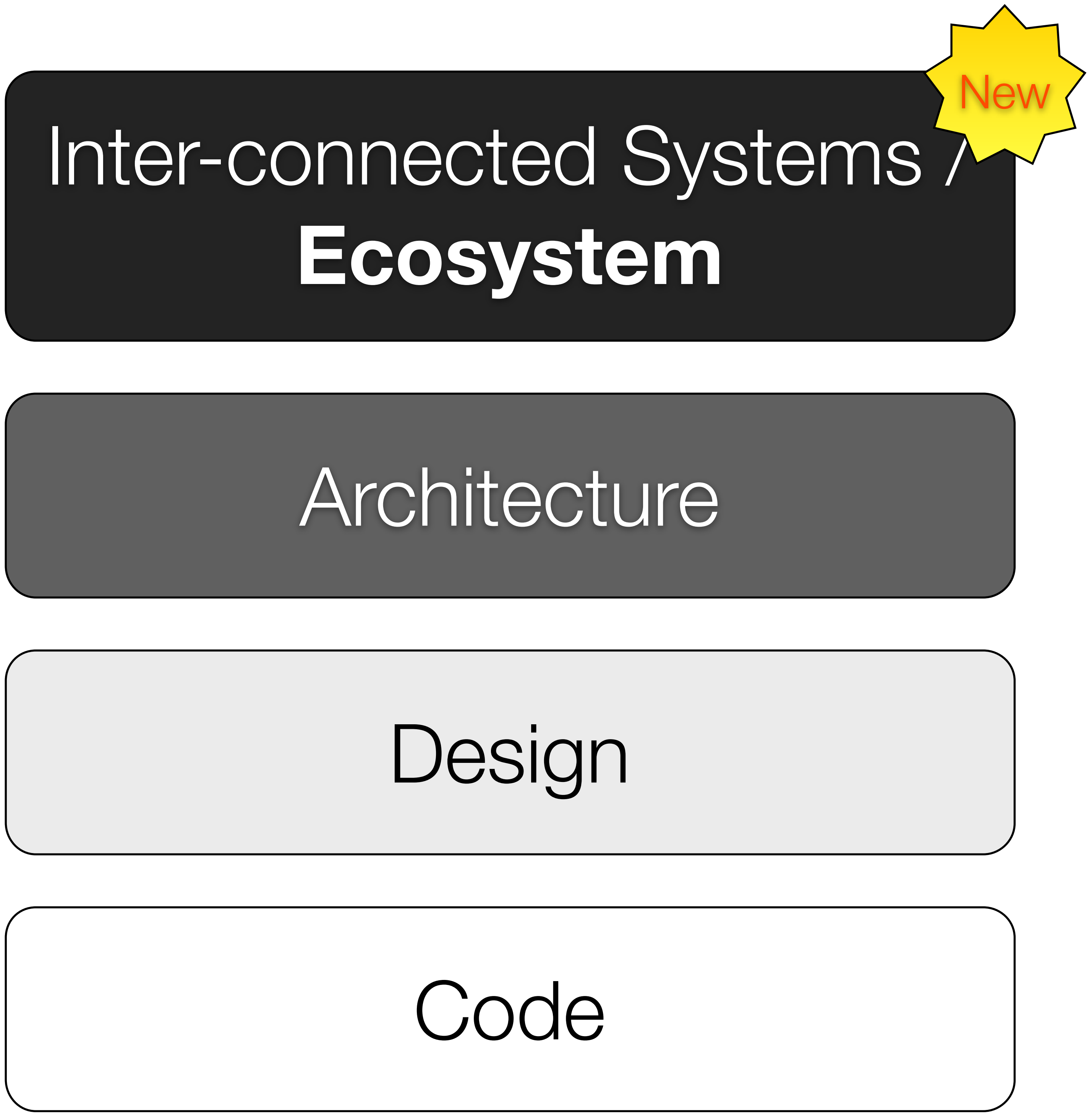
Generators of ecosystems

The 30MLOC of PL/1 Code in a Bank…



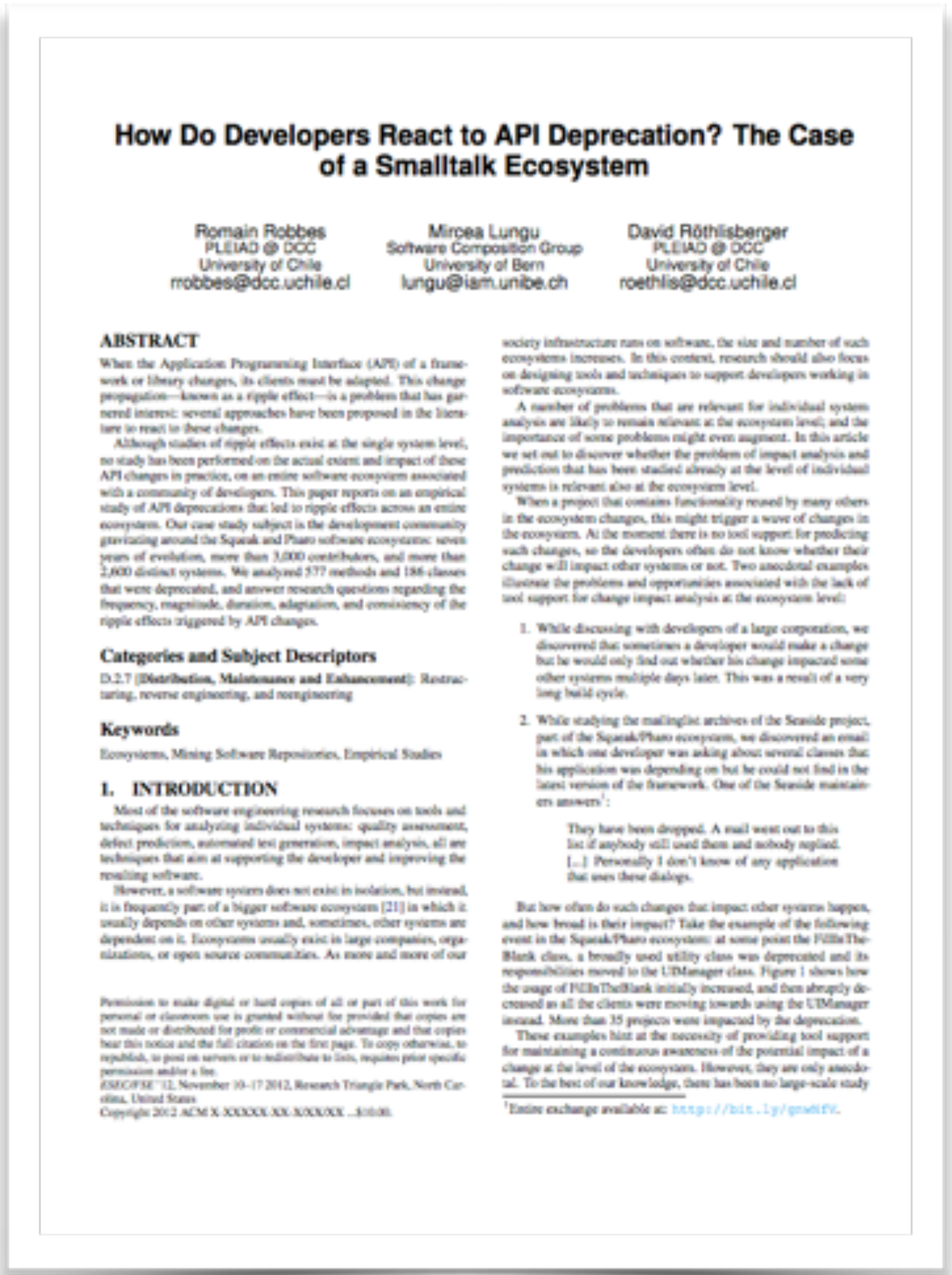
http://scg.unibe.ch/archive/papers/Aesc13a-PL1Ecosystem.pdf

Software Engineering Abstraction Levels



Are software systems in an ecosystem **co-evolving???**

Let us investigate API deprecation!



How Do Developers React to API Deprecation? The Case of a Smalltalk Ecosystem

Romain Robbes
PLEIAD @ DCC
University of Chile
rrobbes@dcc.uchile.cl

Mircea Lungu
Software Composition Group
University of Bern
lungu@iam.unibe.ch

David Röthlisberger
PLEIAD @ DCC
University of Chile
roethlis@dcc.uchile.cl

ABSTRACT

When the Application Programming Interface (API) of a framework or library changes, its clients must be adapted. This change propagation—known as a ripple effect—is a problem that has garnered interest: several approaches have been proposed in the literature to react to these changes.

Although studies of ripple effects exist at the single system level, no study has been performed on the actual extent and impact of these API changes in practice, on an entire software ecosystem associated with a community of developers. This paper reports on an empirical study of API deprecations that led to ripple effects across an entire ecosystem. Our case study subject is the development community gravitating around the Squeak and Pharo software ecosystems: seven years of evolution, more than 3,000 contributors, and more than 2,600 distinct systems. We analyzed 577 methods and 186 classes that were deprecated, and answer research questions regarding the frequency, magnitude, duration, adaptation, and consistency of the ripple effects triggered by API changes.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance and Enhancement**]: Restructuring, reverse engineering, and reengineering

Keywords

Ecosystems, Mining Software Repositories, Empirical Studies

1. INTRODUCTION

Most of the software engineering research focuses on tools and techniques for analyzing individual systems: quality assessment, defect prediction, automated test generation, impact analysis, all are techniques that aim at supporting the developer and improving the resulting software.

However, a software system does not exist in isolation, but instead, it is frequently part of a bigger software ecosystem [21] in which it usually depends on other systems and, sometimes, other systems are dependent on it. Ecosystems usually exist in large companies, organizations, or open source communities. As more and more of our society infrastructure runs on software, the size and number of such ecosystems increases. In this context, research should also focus on designing tools and techniques to support developers working in software ecosystems.

A number of problems that are relevant for individual system analysis are likely to remain relevant at the ecosystem level; and the importance of some problems might even augment. In this article we set out to discover whether the problem of impact analysis and prediction that has been studied already at the level of individual systems is relevant also at the ecosystem level.

When a project that contains functionality reused by many others in the ecosystem changes, this might trigger a wave of changes in the ecosystem. At the moment there is no tool support for predicting such changes, so the developers often do not know whether their change will impact other systems or not. Two anecdotal examples illustrate the problems and opportunities associated with the lack of tool support for change impact analysis at the ecosystem level:

1. While discussing with developers of a large corporation, we discovered that sometimes a developer would make a change but he would only find out whether his change impacted some other systems multiple days later. This was a result of a very long build cycle.

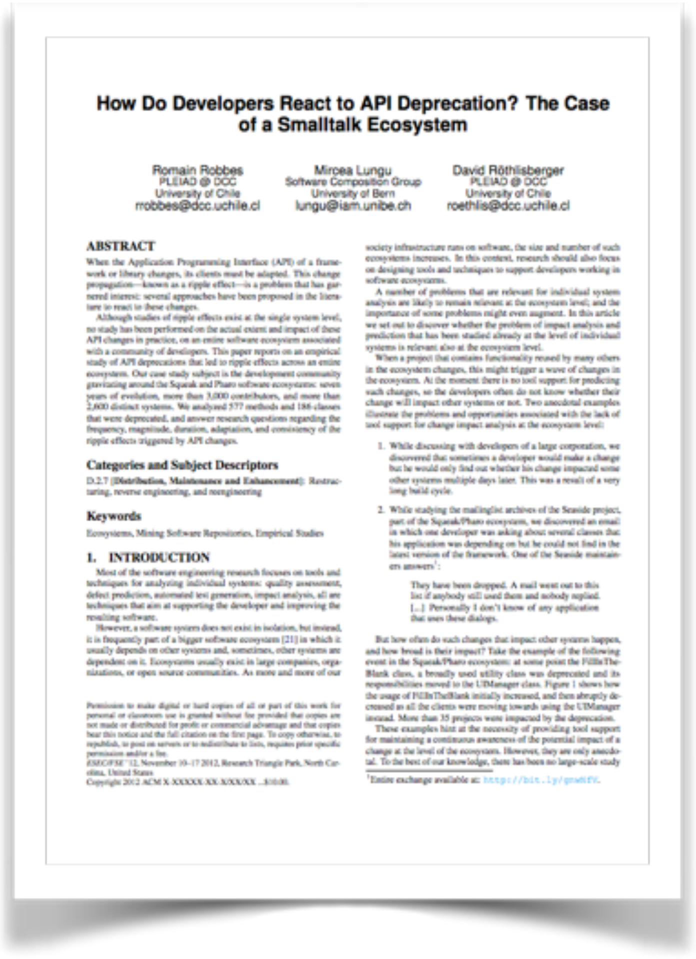
2. While studying the mailinglist archives of the Seaside project, part of the Squeak/Pharo ecosystem, we discovered an email in which one developer was asking about several classes that his application was depending on but he could not find in the latest version of the framework. One of the Seaside maintainers answers[1]:

> 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.

But how often do such changes that impact other systems happen, and how broad is their impact? Take the example of the following event in the Squeak/Pharo ecosystem: at some point the FillInTheBlank class, a broadly used utility class was deprecated and its responsibilities moved to the UIManager class. Figure 1 shows how the usage of FillInTheBlank initially increased, and then abruptly decreased as all the clients were moving towards using the UIManager instead. More than 35 projects were impacted by the deprecation.

These examples hint at the necessity of providing tool support for maintaining a continuous awareness of the potential impact of a change at the level of the ecosystem. However, they are only anecdotal. To the best of our knowledge, there has been no large-scale study

[1]Entire exchange available at: http://bit.ly/gnw6fV.



How Do Developers React to API Deprecation? The Case of a Smalltalk Ecosystem

Romain Robbes, PLEIAD @ DCC, University of Chile, rrobbes@dcc.uchile.cl
Mircea Lungu, Software Composition Group, University of Bern, lungu@iam.unibe.ch
David Röthlisberger, PLEIAD @ DCC, University of Chile, roethlis@dcc.uchile.cl

ABSTRACT

When the Application Programming Interface (API) of a framework or library changes, its clients must be adapted. This change propagation—known as a ripple effect—is a problem that has garnered interest: several approaches have been proposed in the literature to react to these changes.

Although studies of ripple effects exist at the single system level, no study has been performed on the actual extent and impact of these API changes in practice, on an entire software ecosystem associated with a community of developers. This paper reports on an empirical study of API deprecations that led to ripple effects across an entire ecosystem. Our case study subject is the development community gravitating around the Squeak and Pharo software ecosystems: seven years of evolution, more than 3,000 contributors, and more than 2,600 distinct systems. We analyzed 577 methods and 186 classes that were deprecated, and answer research questions regarding the frequency, magnitude, duration, adaptation, and consistency of the ripple effects triggered by API changes.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance and Enhancement**]: Restructuring, reverse engineering, and reengineering

Keywords

Ecosystems, Mining Software Repositories, Empirical Studies

1. INTRODUCTION

Most of the software engineering research focuses on tools and techniques for analyzing individual systems: quality assessment, defect prediction, automated test generation, impact analysis, all are techniques that aim at supporting the developer and improving the resulting software.

However, a software system does not exist in isolation, but instead, it is frequently part of a bigger software ecosystem [21] in which it usually depends on other systems and, sometimes, other systems are dependent on it. Ecosystems usually exist in large companies, organizations, or open source communities. As more and more of our society infrastructure runs on software, the size and number of such ecosystems increases. In this context, research should also focus on designing tools and techniques to support developers working in software ecosystems.

A number of problems that are relevant for individual system analysis are likely to remain relevant at the ecosystem level; and the importance of some problems might even augment. In this article we set out to discover whether the problem of impact analysis and prediction that has been studied already at the level of individual systems is relevant also at the ecosystem level.

When a project that contains functionality reused by many others in the ecosystem changes, this might trigger a wave of changes in the ecosystem. At the moment there is no tool support for predicting such changes, so the developers often do not know whether their change will impact other systems or not. Two anecdotal examples illustrate the problems and opportunities associated with the lack of tool support for change impact analysis at the ecosystem level:

1. While discussing with developers of a large corporation, we discovered that sometimes a developer would make a change but he would only find out whether his change impacted some other systems multiple days later. This was a result of a very long build cycle.
2. While studying the mailinglist archives of the Seaside project, part of the Squeak/Pharo ecosystem, we discovered an email in which one developer was asking about several classes that his application was depending on but he could not find in the latest version of the framework. One of the Seaside maintainers answers[1]:

   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.

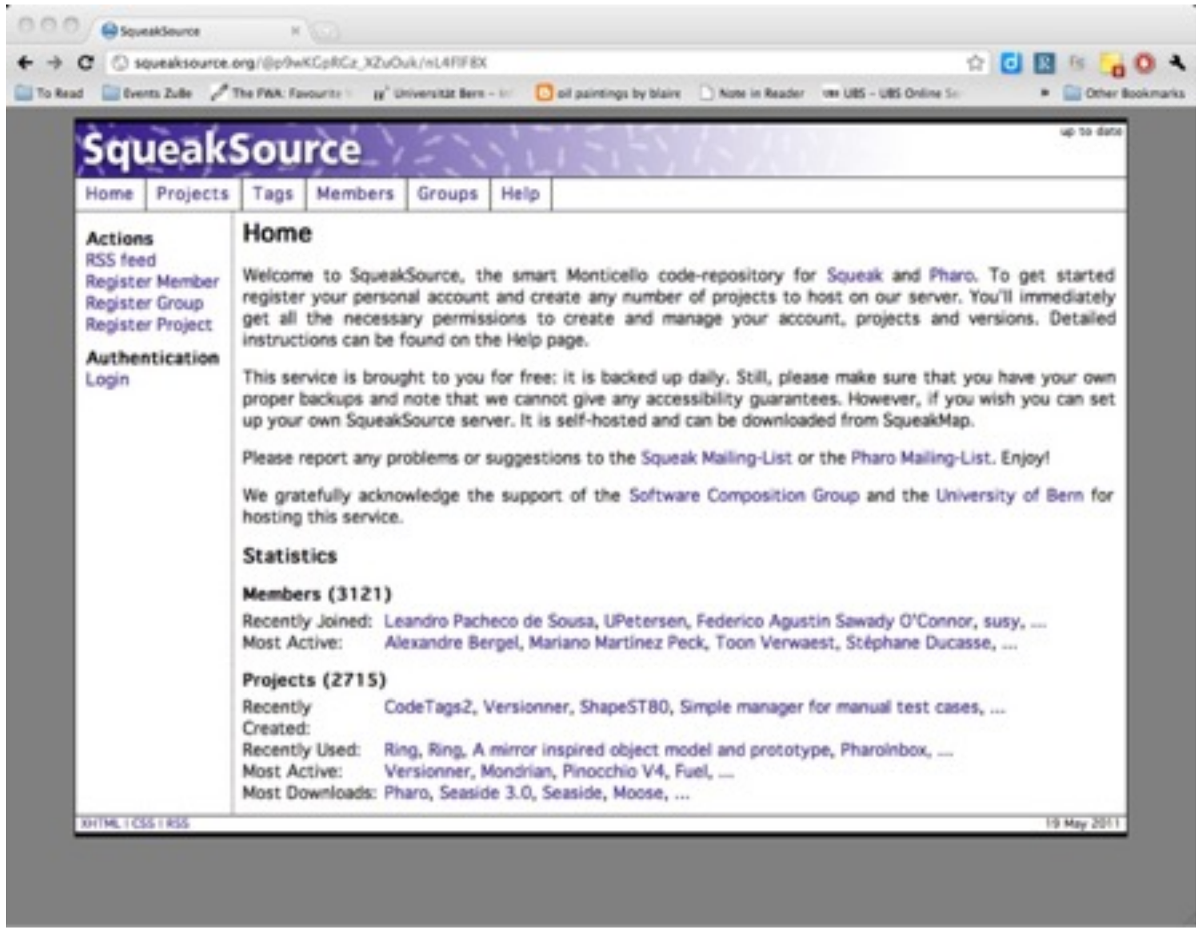
But how often do such changes that impact other systems happen, and how broad is their impact? Take the example of the following event in the Squeak/Pharo ecosystem: at some point the FillInTheBlank class, a broadly used utility class was deprecated and its responsibilities moved to the UIManager class. Figure 1 shows how the usage of FillInTheBlank initially increased, and then abruptly decreased as all the clients were moving towards using the UIManager instead. More than 35 projects were impacted by the deprecation.

These examples hint at the necessity of providing tool support for maintaining a continuous awareness of the potential impact of a change at the level of the ecosystem. However, they are only anecdotal. To the best of our knowledge, there has been no large-scale study

ESEC/FSE'12, November 10-17 2012, Research Triangle Park, North Carolina, United States
Copyright 2012 ACM X-XXXXX-XX-X/XX/XX ...$10.00.

[1]Entire exchange available at: http://bit.ly/gnwRfV.

8 years



Home

Welcome to SqueakSource, the smart Monticello code-repository for Squeak and Pharo. To get started register your personal account and create any number of projects to host on our server. You'll immediately get all the necessary permissions to create and manage your account, projects and versions. Detailed instructions can be found on the Help page.

This service is brought to you for free: it is backed up daily. Still, please make sure that you have your own proper backups and note that we cannot give any accessibility guarantees. However, if you wish you can set up your own SqueakSource server. It is self-hosted and can be downloaded from SqueakMap.

Please report any problems or suggestions to the Squeak Mailing-List or the Pharo Mailing-List. Enjoy!

We gratefully acknowledge the support of the Software Composition Group and the University of Bern for hosting this service.

Statistics

Members (3121)

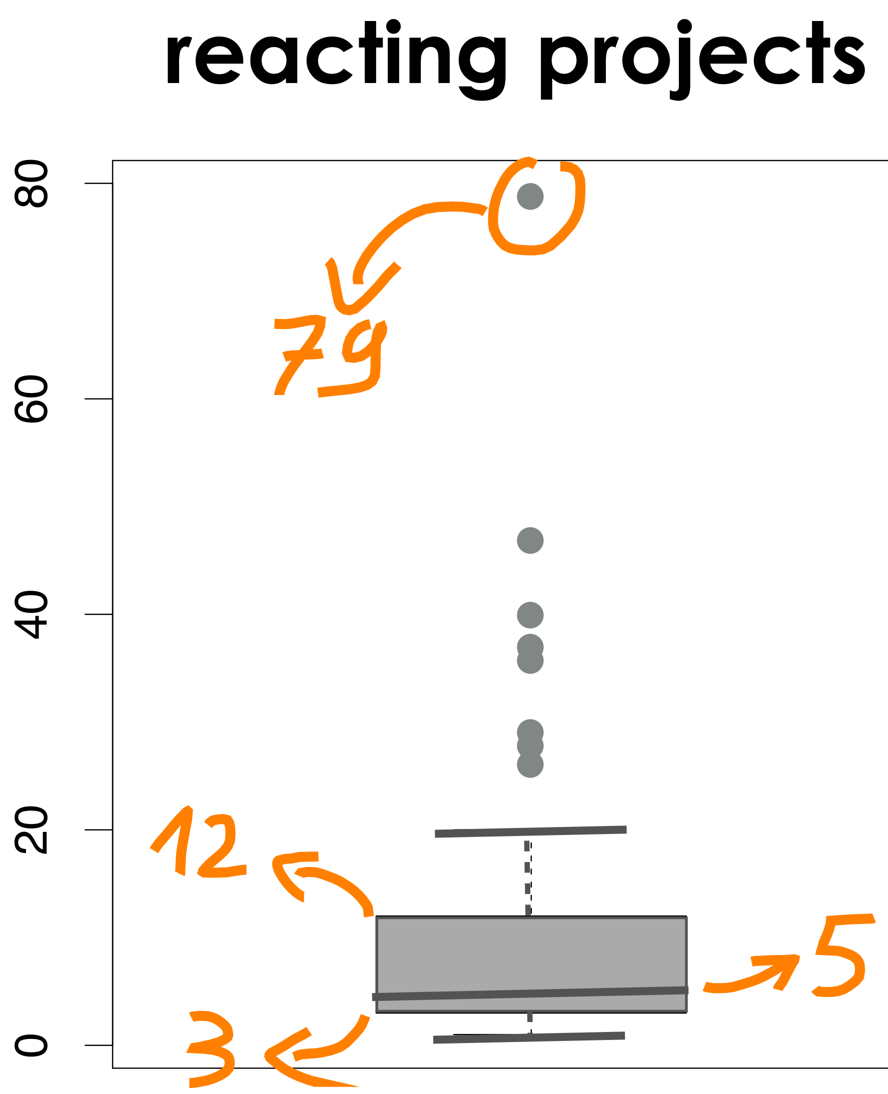
Recently Joined: Leandro Pacheco de Sousa, UPetersen, Federico Agustin Sawady O'Connor, susy, ...
Most Active: Alexandre Bergel, Mariano Martinez Peck, Toon Verwaest, Stéphane Ducasse, ...

Projects (2715)

Recently Created: CodeTags2, Versionner, ShapeST80, Simple manager for manual test cases, ...
Recently Used: Ring, Ring, A mirror inspired object model and prototype, PharoInbox, ...
Most Active: Versionner, Mondrian, Pinocchio V4, Fuel, ...
Most Downloads: Pharo, Seaside 3.0, Seaside, Moose, ...

Contributors:	**2.300**
Projects:	**2.500**
Classes:	**95.000**
Commits:	**110.000**
LOC:	**600.000.000**

RQ$_2$: Magnitude of ripple effects triggered by API deprecation?

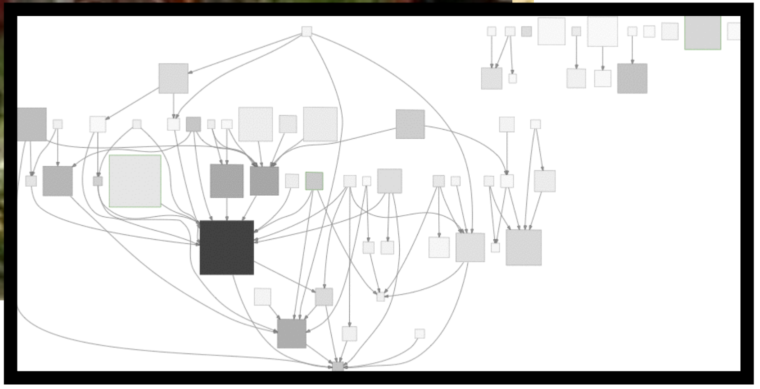


- deprecations regularly impact the ecosystem
- the impact can be large

True Story

Why is my code broken today when it worked yesterday?

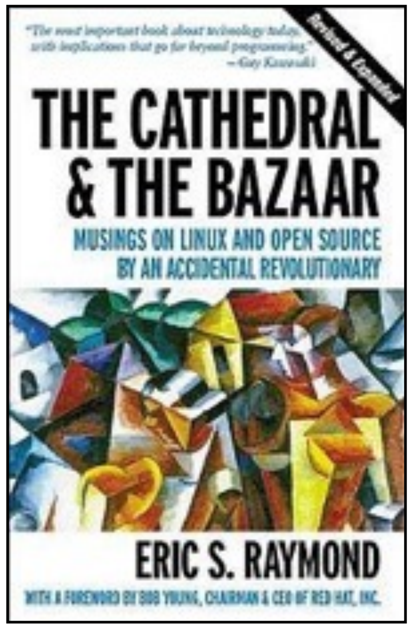
Opportunities



Standing on the shoulders of giants

Good programmers know what to write; great ones know what to rewrite and reuse.

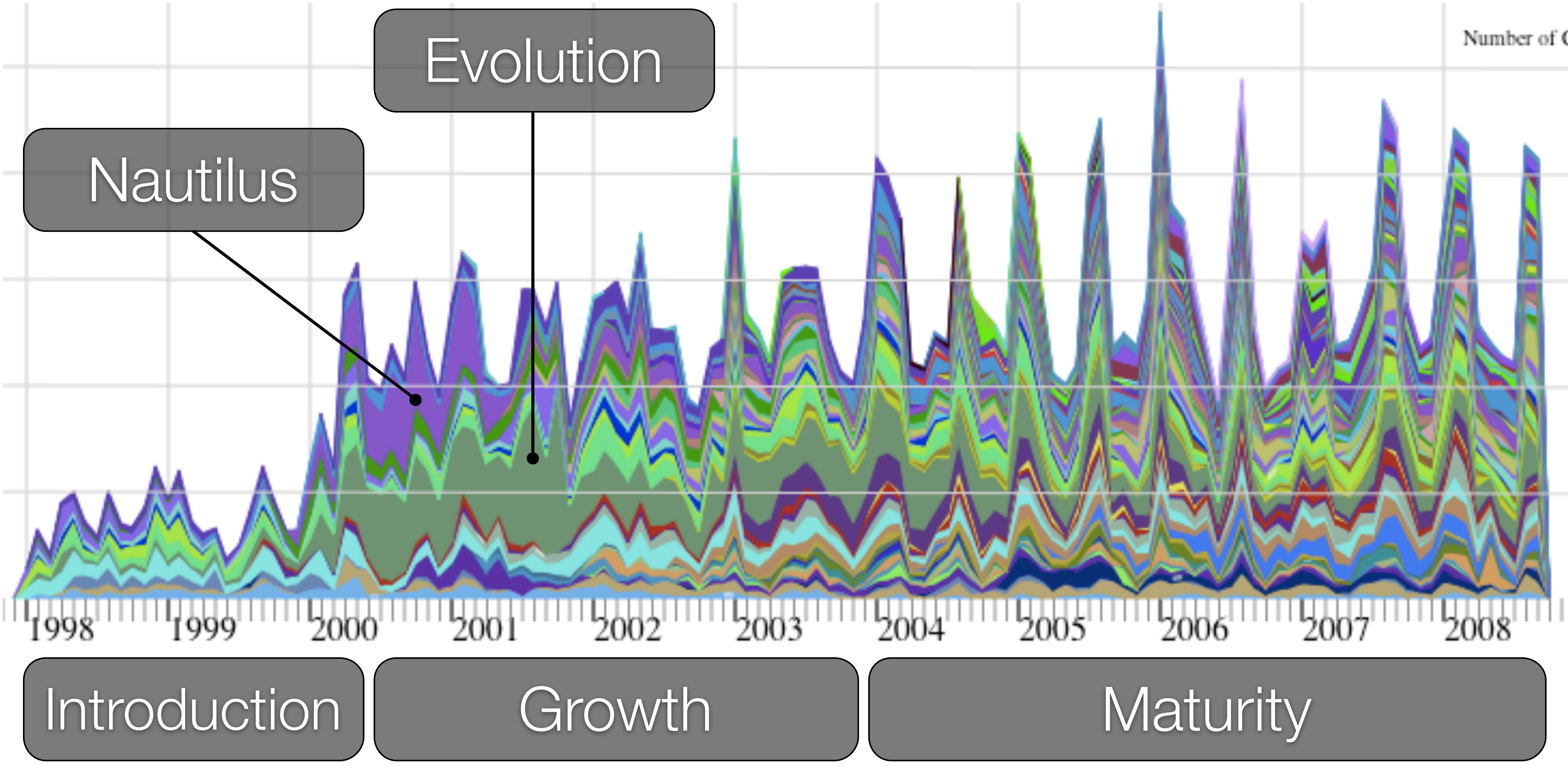
(ES Raymond, The Cathedral and The Bazaar)


THE CATHEDRAL
& THE BAZAAR
MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY
ERIC S. RAYMOND
WITH A FOREWORD BY BOB YOUNG, CHAIRMAN & CEO OF RED HAT, INC.

The Network Effect

The value of an ecosystem increases with the number of systems it contains

The Evolution of Gnome



Better Tool and Developer Support

Mining Trends in Library Usage

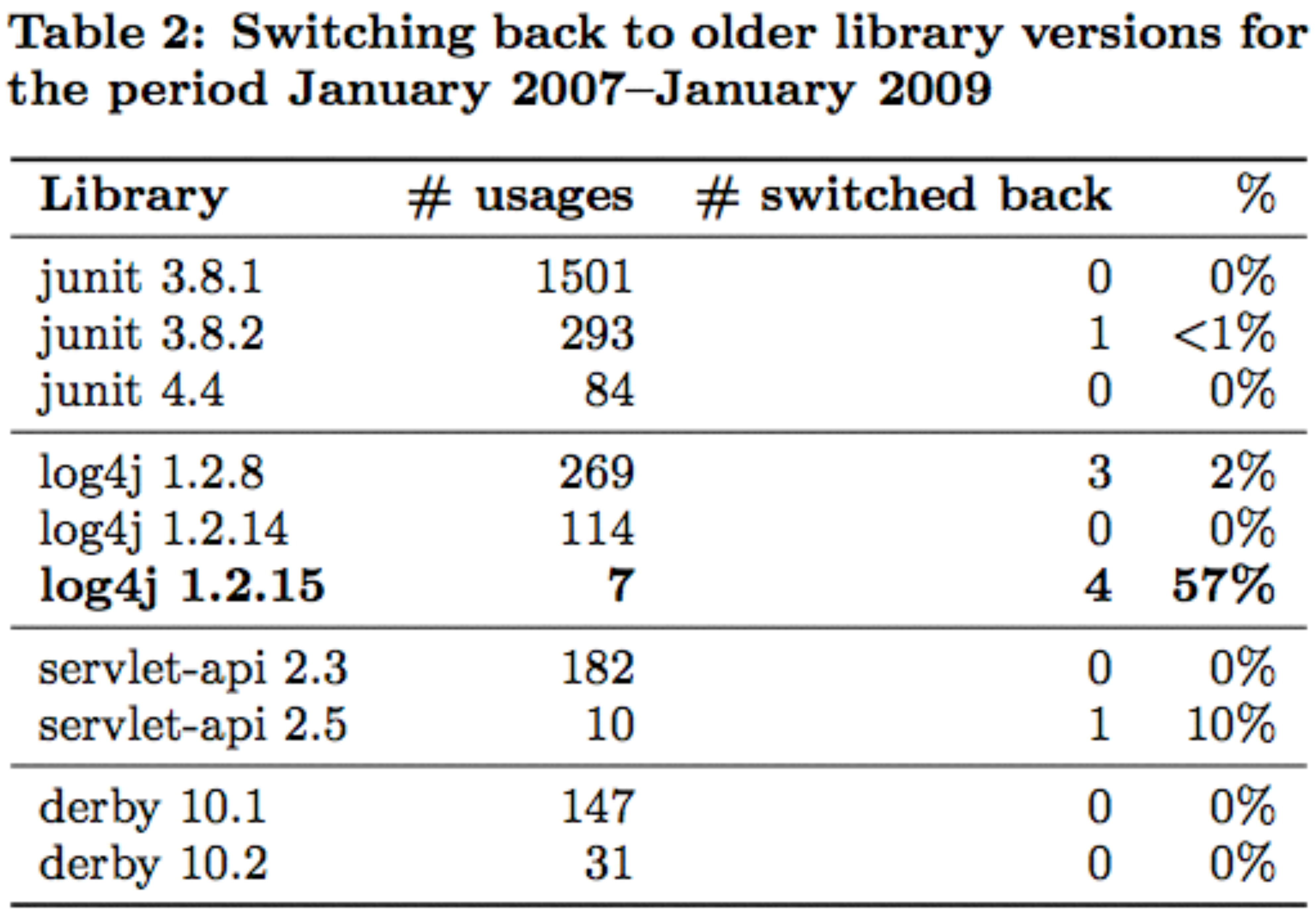
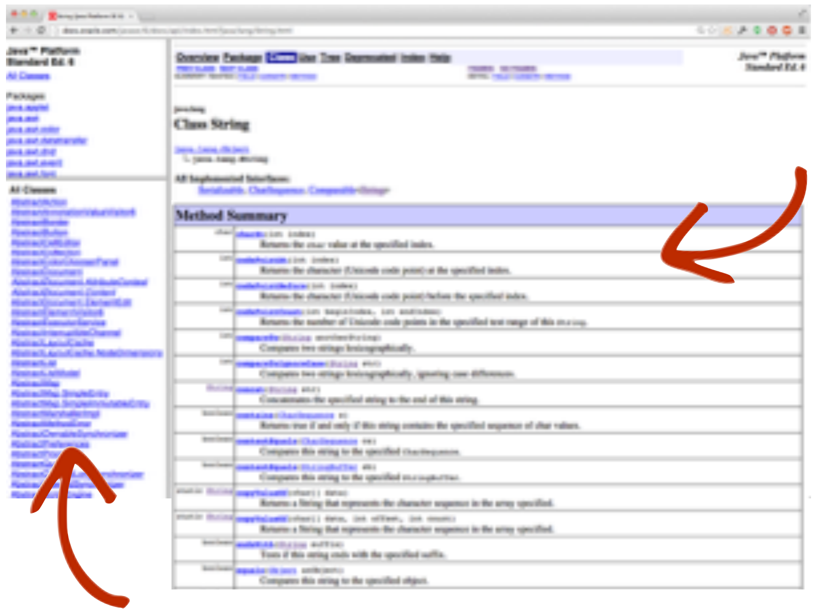


Table 2: Switching back to older library versions for the period January 2007–January 2009

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%

“Alphabetical ordering must die.”

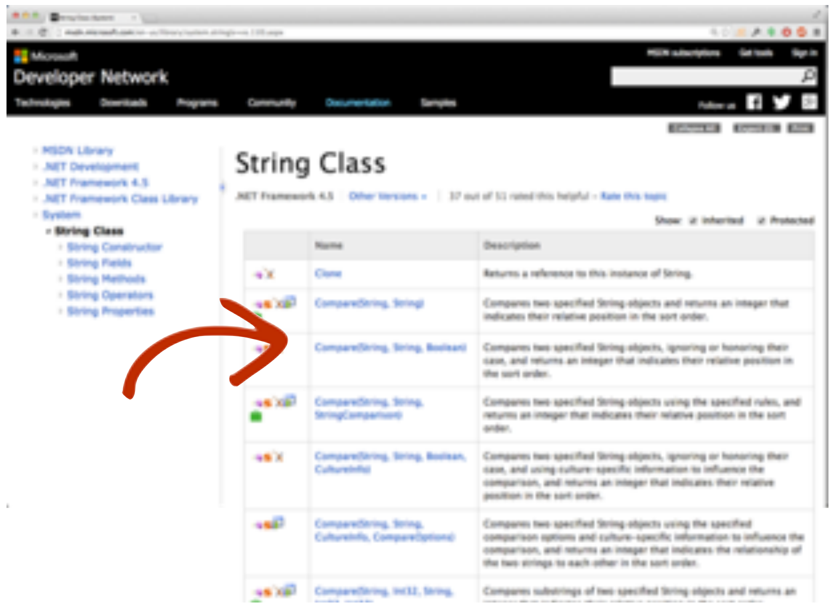
–Jakob Nielsen



Class String

Method Summary

JavaDoc



Developer Network

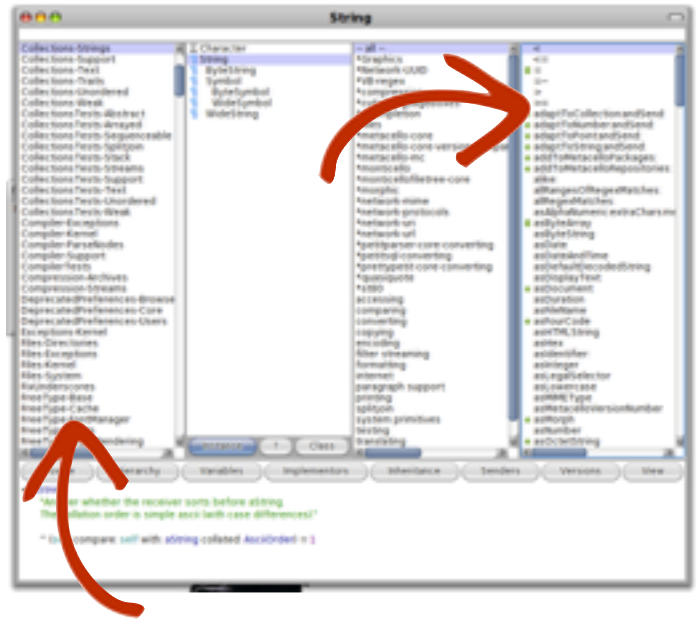
- MSDN Library
- .NET Development
- .NET Framework 4.5
- .NET Framework Class Library
- System
- String Class
  - String Constructor
  - String Fields
  - String Methods
  - String Operators
  - String Properties

String Class

.NET Framework 4.5

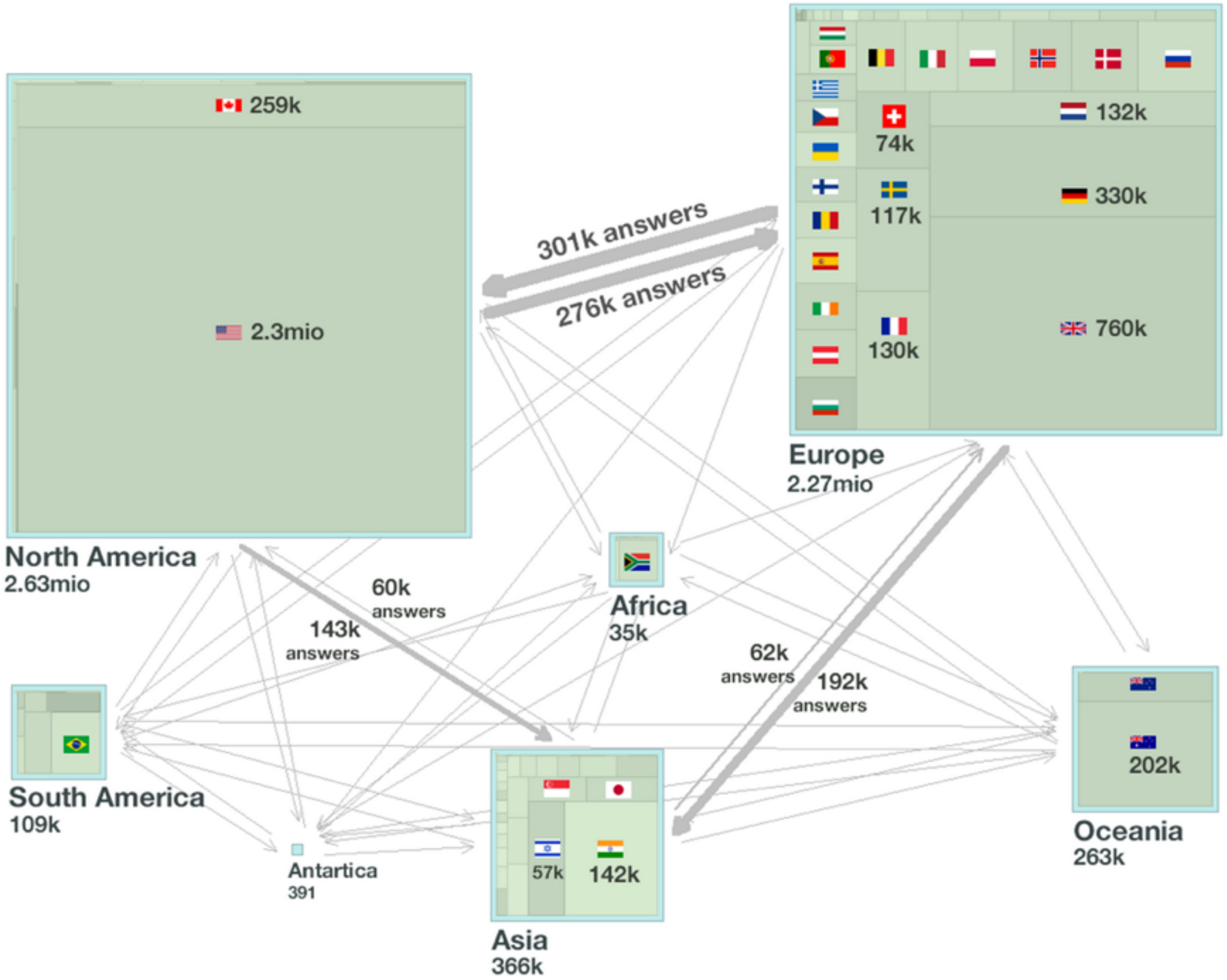
	Name	Description
	Clone	Returns a reference to this instance of String.
	Compare(String, String)	Compares two specified String objects and returns an integer that indicates their relative position in the sort order.
	Compare(String, String, Boolean)	Compares two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.
	Compare(String, String, StringComparison)	Compares two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.
	Compare(String, String, Boolean, CultureInfo)	Compares two specified String objects, ignoring or honoring their case, and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.
	Compare(String, String, CultureInfo, CompareOptions)	Compares two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two strings to each other in the sort order.

MSDN



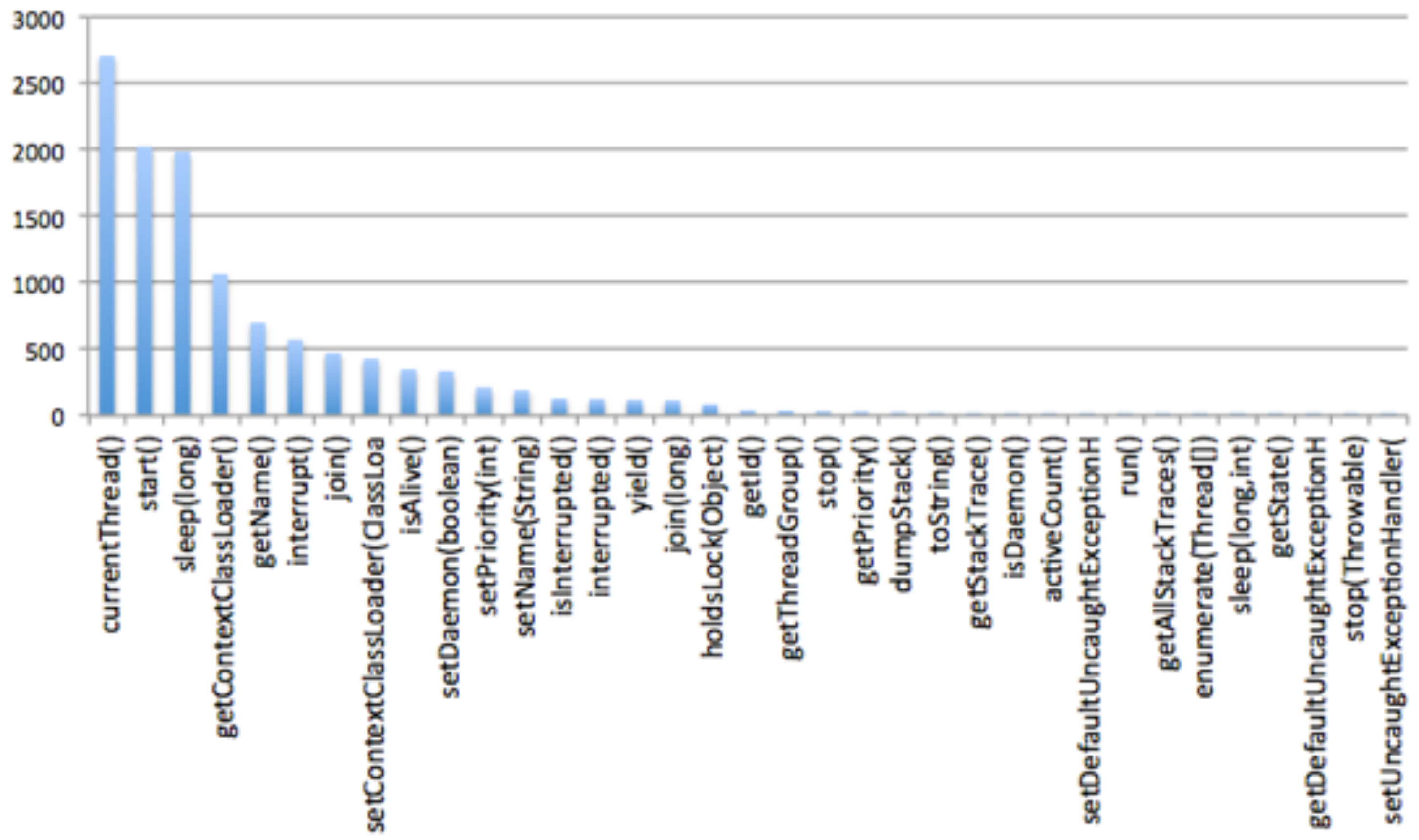
String

Nautilius



Geo-locating Knowledge Transfer in StackOverflow
Schenk, Lungu. SSE 2013

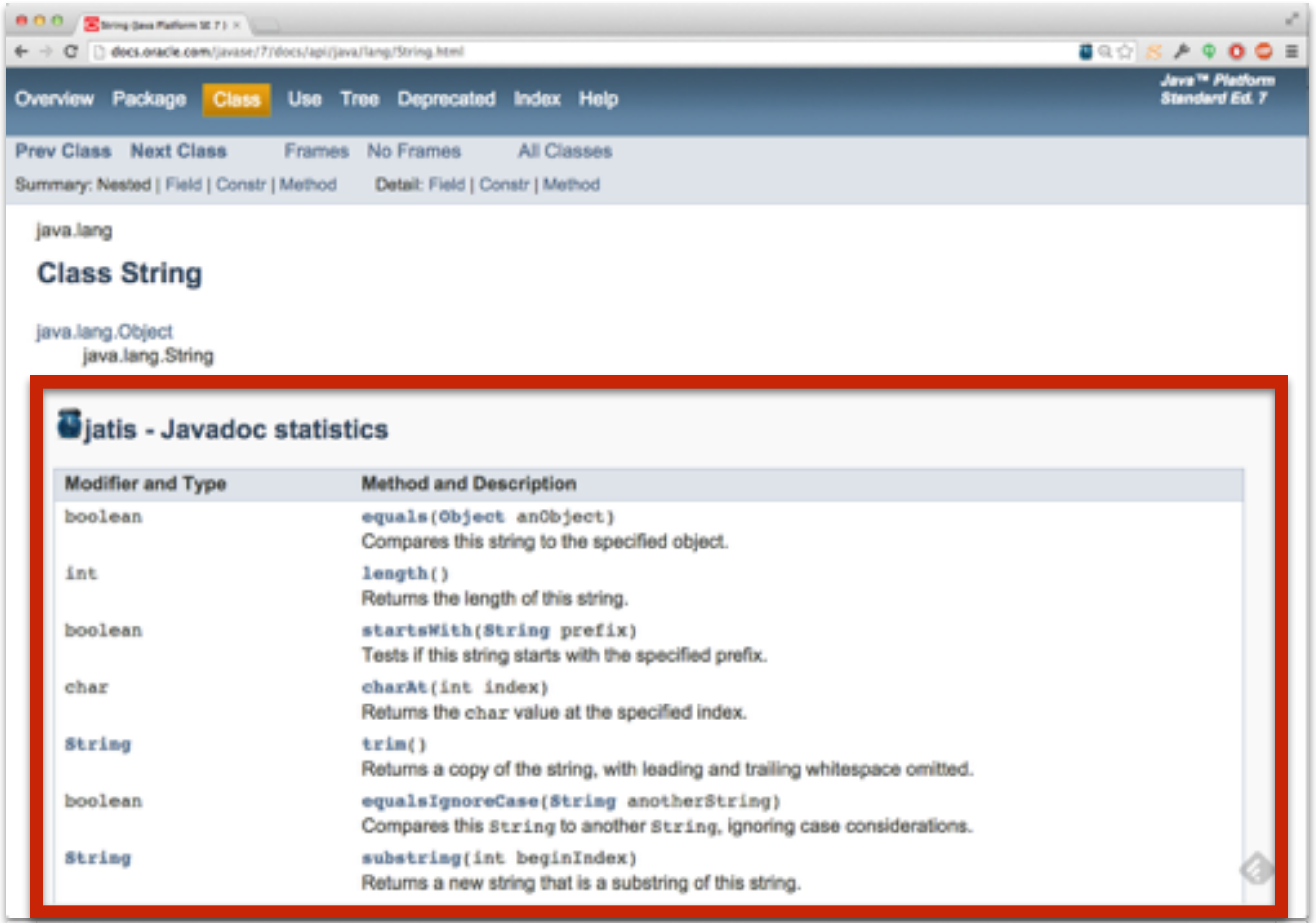
Usage of java.lang.Thread API in the Java Ecosystem



https://github.com/caracciolo/pangea/blob/master/demo/thread-stats.st

data mining downstream usage patterns

JavaDoc



Overview
Package
Class
Use
Tree
Deprecated
Index
Help
Java™ Platform Standard Ed. 7
Prev Class
Next Class
Frames
No Frames
All Classes
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method

java.lang

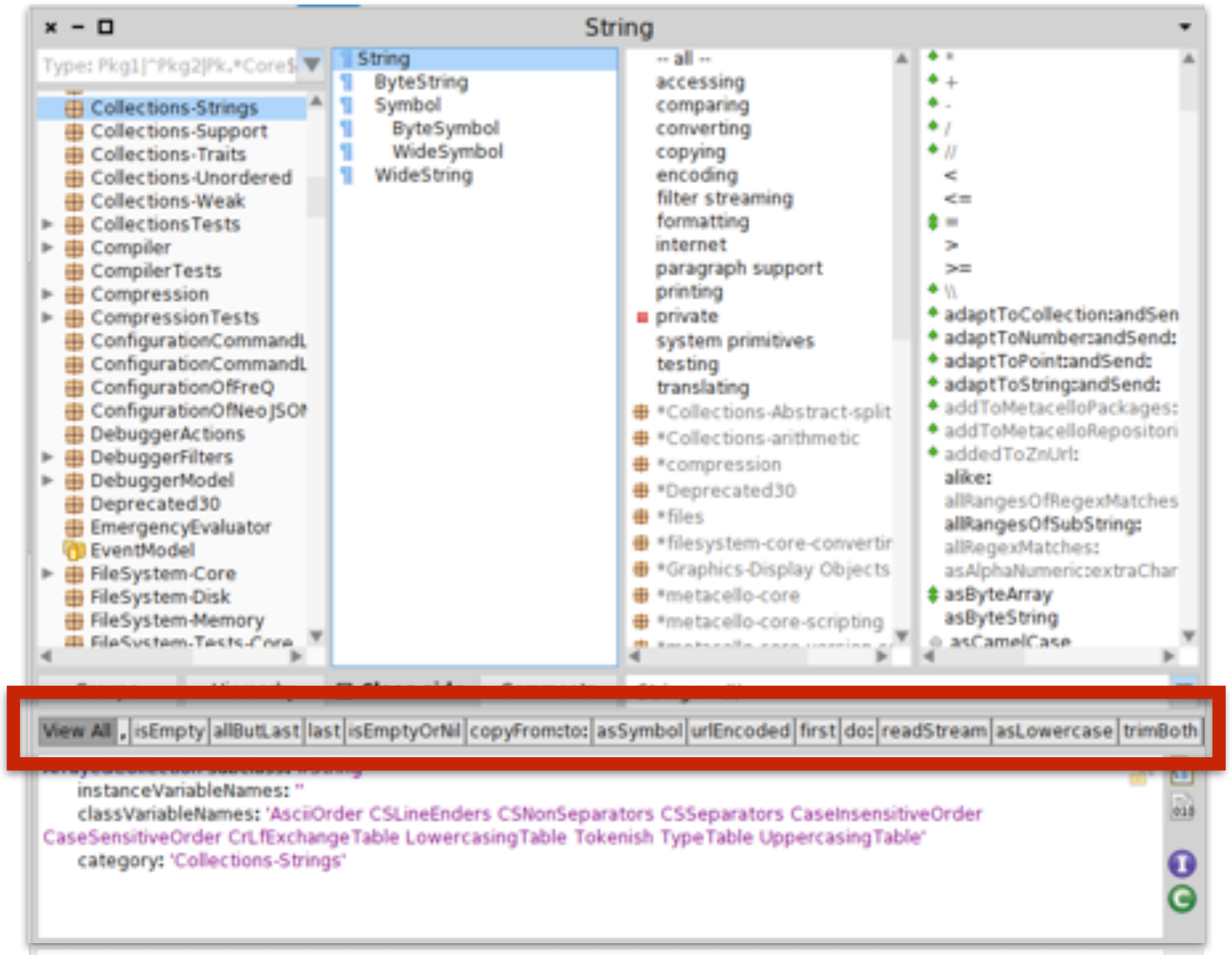
Class String

java.lang.Object
    java.lang.String

jatis - Javadoc statistics

Modifier and Type	Method and Description
`boolean`	`equals(Object anObject)` Compares this string to the specified object.
`int`	`length()` Returns the length of this string.
`boolean`	`startsWith(String prefix)` Tests if this string starts with the specified prefix.
`char`	`charAt(int index)` Returns the `char` value at the specified index.
`String`	`trim()` Returns a copy of the string, with leading and trailing whitespace omitted.
`boolean`	`equalsIgnoreCase(String anotherString)` Compares this `String` to another `String`, ignoring case considerations.
`String`	`substring(int beginIndex)` Returns a new string that is a substring of this string.

Nautilius

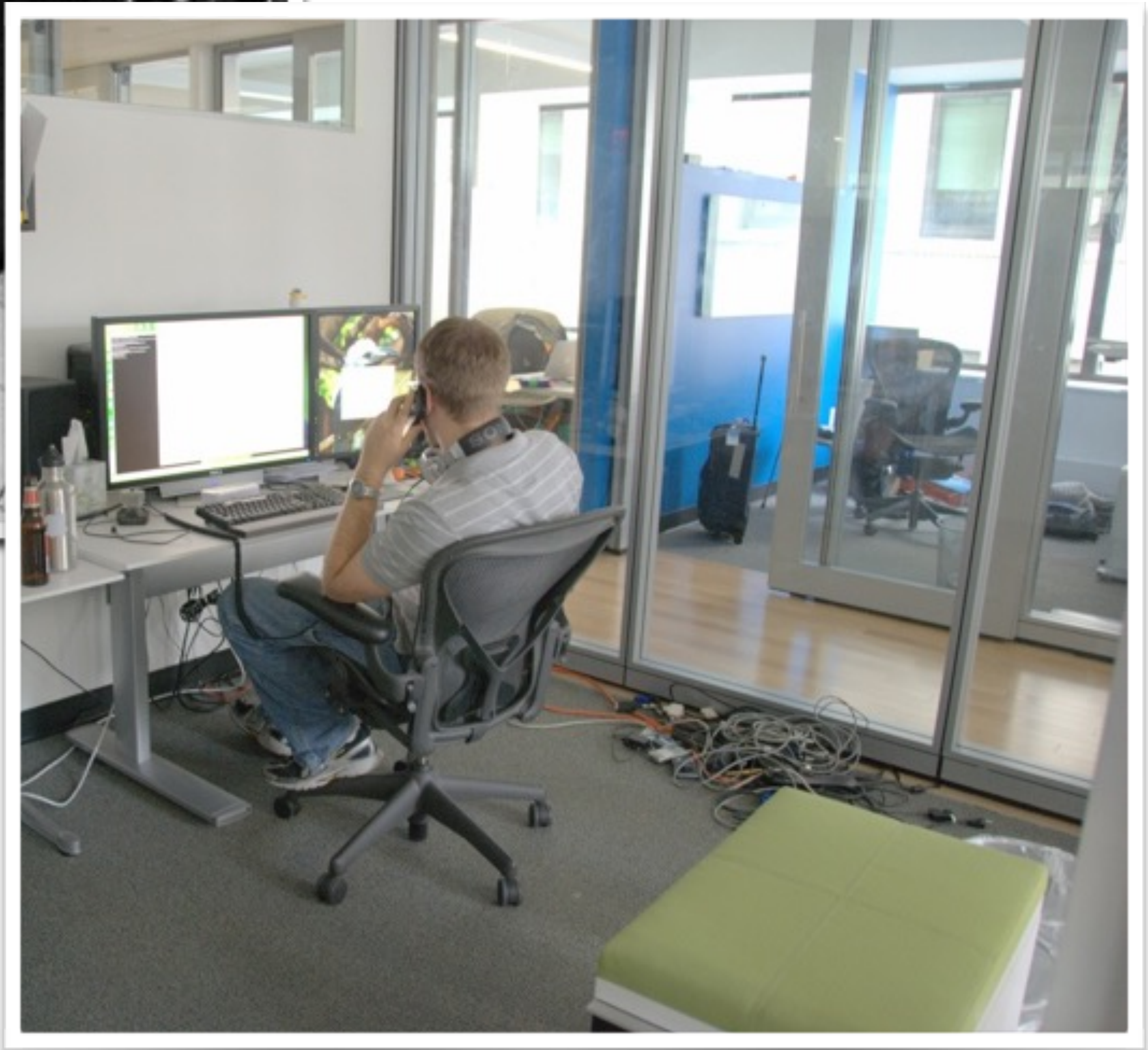


```
	instanceVariableNames: ''
	classVariableNames: 'AsciiOrder CSLineEnders CSNonSeparators CSSeparators CaseInsensitiveOrder
CaseSensitiveOrder CrLfExchangeTable LowercasingTable Tokenish TypeTable UppercasingTable'
	category: 'Collections-Strings'
```

Challenges

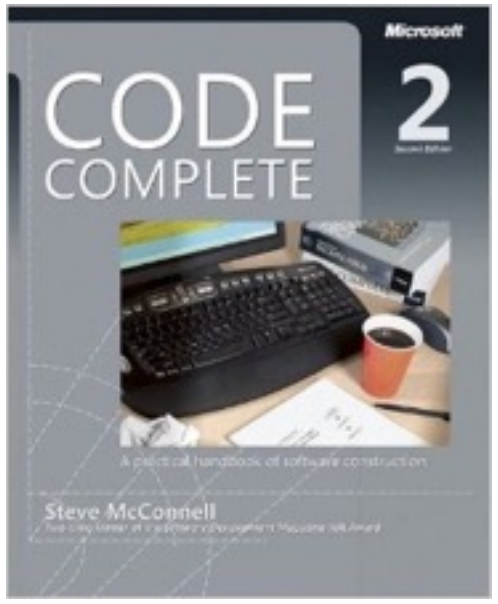
Scale

Makes everything slower



Things that are affected:

build times, testing times, understanding,. etc.

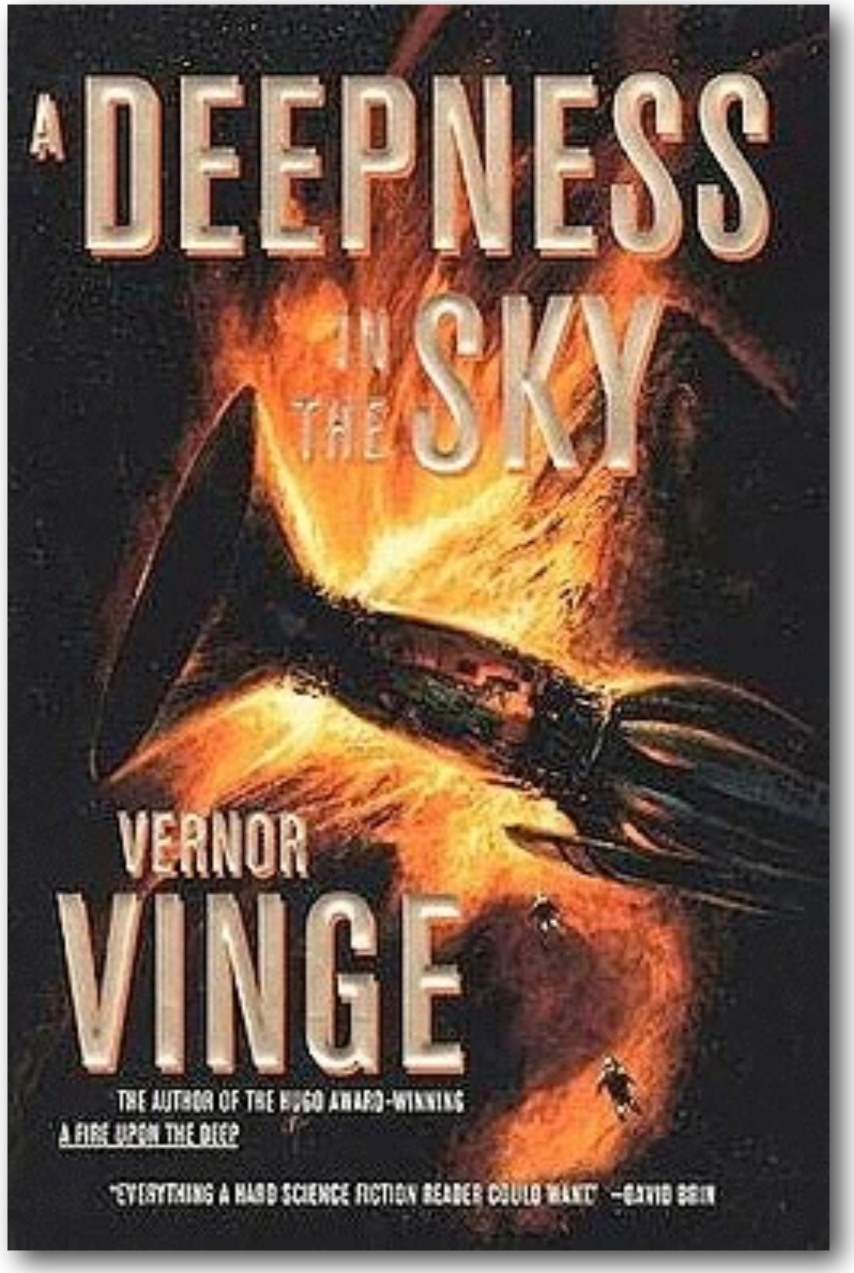


The industry-average productivity for a software product is about 10 to 50 of lines of delivered code per person per day (including all non-coding overhead).

Chapter 20.5

Trust

Not upset that you lied to me.
Upset that from now on I can't believe you.


A DEEPNESS IN THE SKY
VERNOR VINGE
THE AUTHOR OF THE HUGO AWARD-WINNING A FIRE UPON THE DEEP
"EVERYTHING A HARD SCIENCE FICTION READER COULD WANT" —DAVID BRIN

"When systems depended on underlying systems, and those depended on things still older... it became impossible to know all the systems could do"

The Law of Leaky Abstractions

“All non-trivial abstractions, to some degree, are leaky”.

J. Spolsky

SQL
Database Language

Object1
foo()
Object2
foo()
ORM

Designing a Run-Time Ecosystem…

How to encourage interaction
while minimizing the required trust?

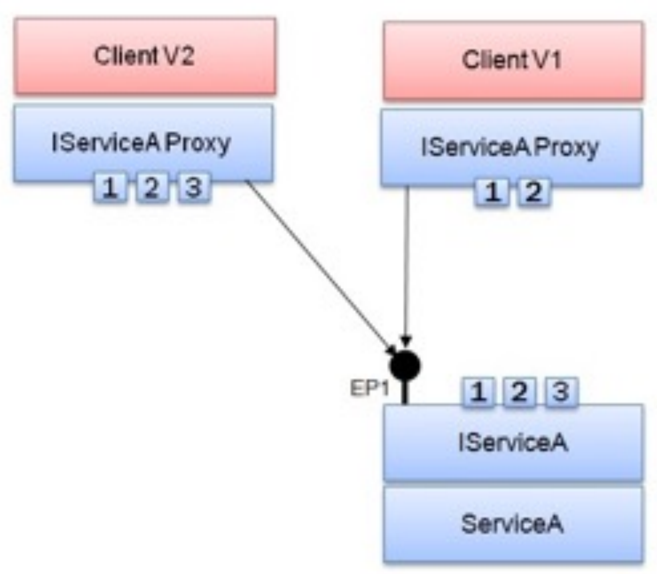
https://developer.chrome.com/extensions/content_scripts

http://developer.android.com/guide/components/intents-filters.html

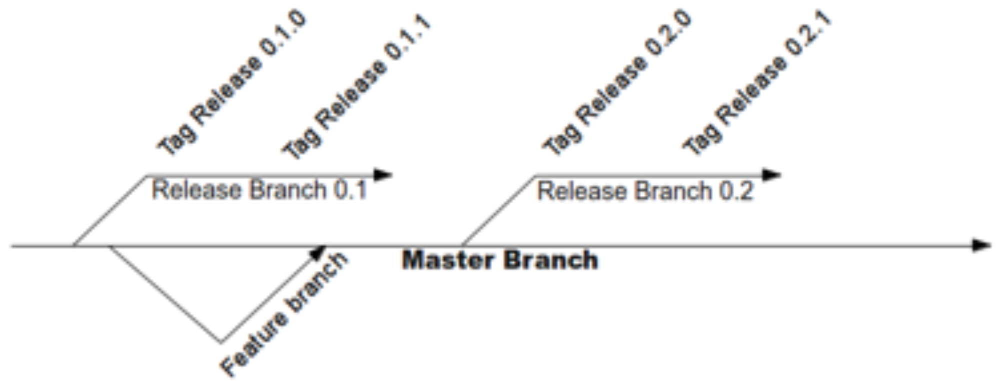
Interdependence

Is a mixed blessing

Controlling the upstream propagation



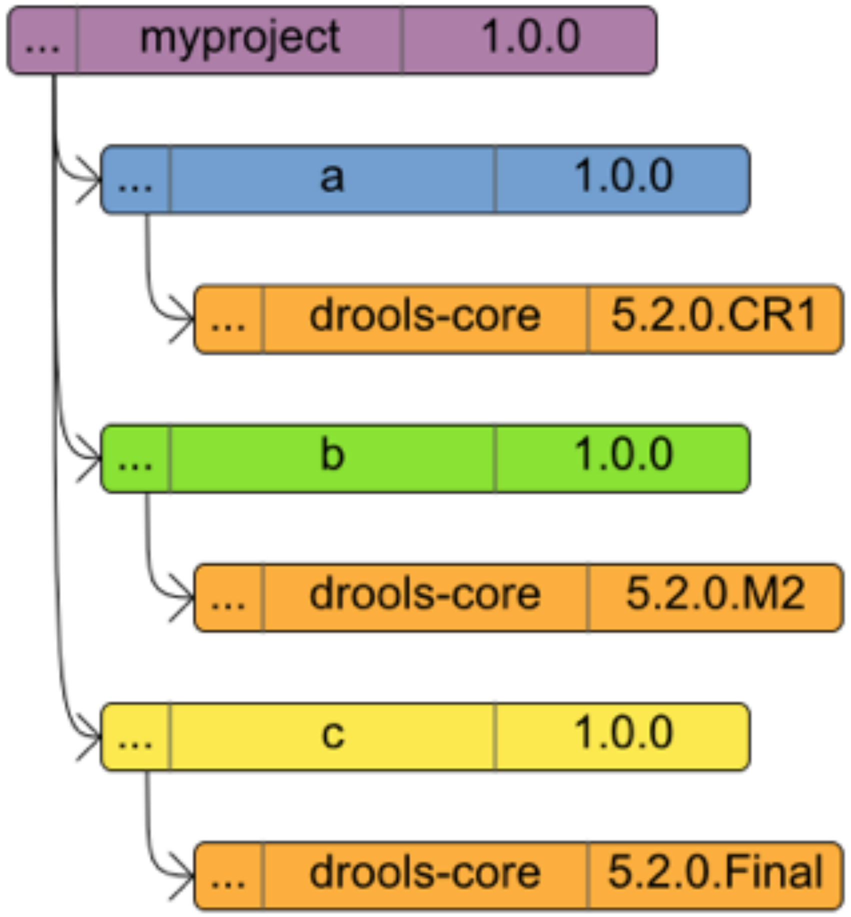
run-time



compile-time

Projects need to isolate themselves from the evolution of the others

Dependency Hell



1. Conflicting dependencies

2. Long chains of dependencies

3. Large Number of Dependencies

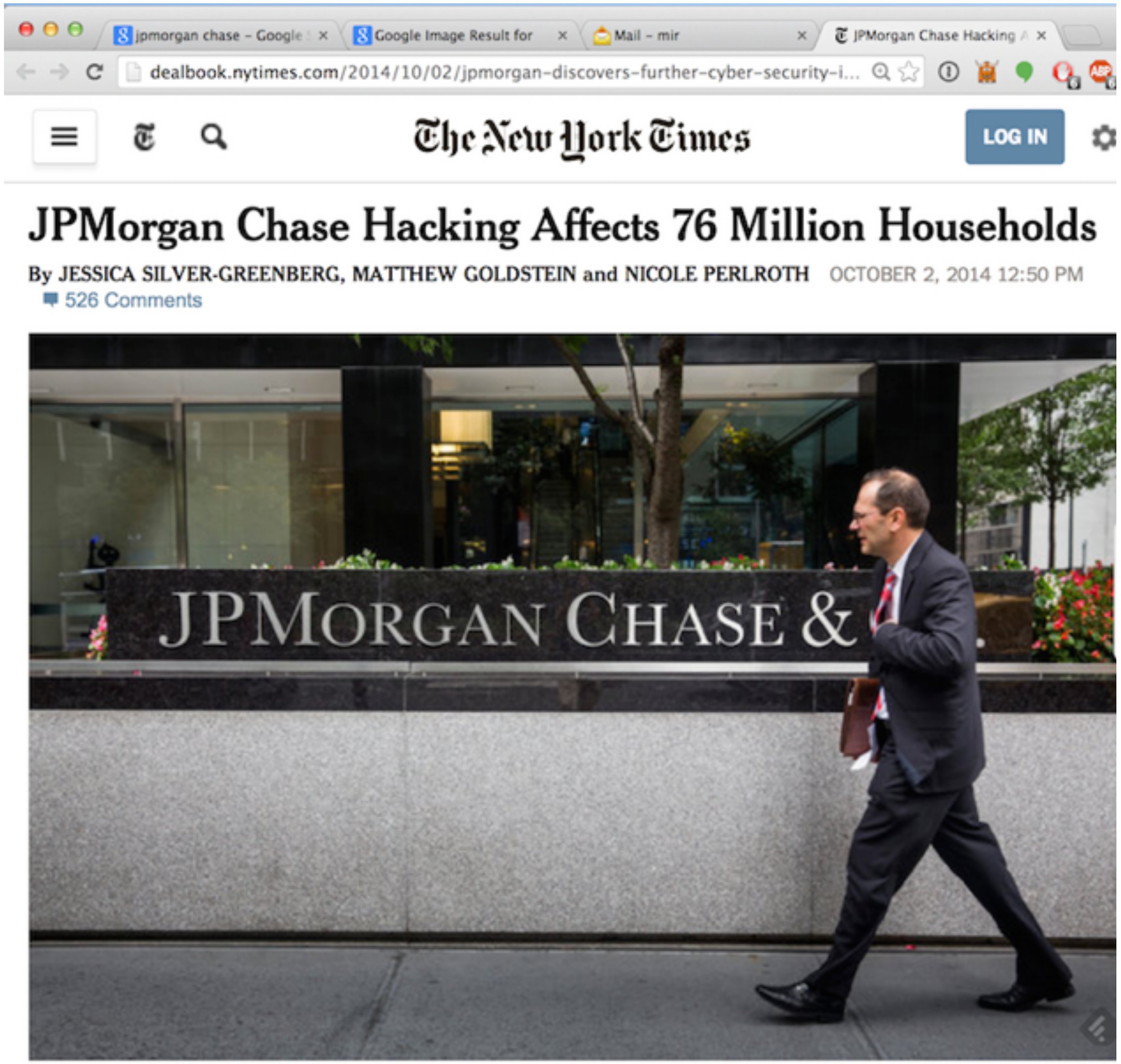
More Downsides of Inter-dependency…

Upstream evolution can be used as a strategy

Co-evolution can take a lot of effort. Must plan for co-evolution and put time aside.
(The Importance of Slack)

Awareness

Gets more difficult



JPMorgan Chase Hacking Affects 76 Million Households

By JESSICA SILVER-GREENBERG, MATTHEW GOLDSTEIN and NICOLE PERLROTH OCTOBER 2, 2014 12:50 PM

526 Comments

Keeping up with the upstream is challenging

Survey of Information Needs in Microsoft

Find the relevant engineers for a feature

Find an expert on a given feature

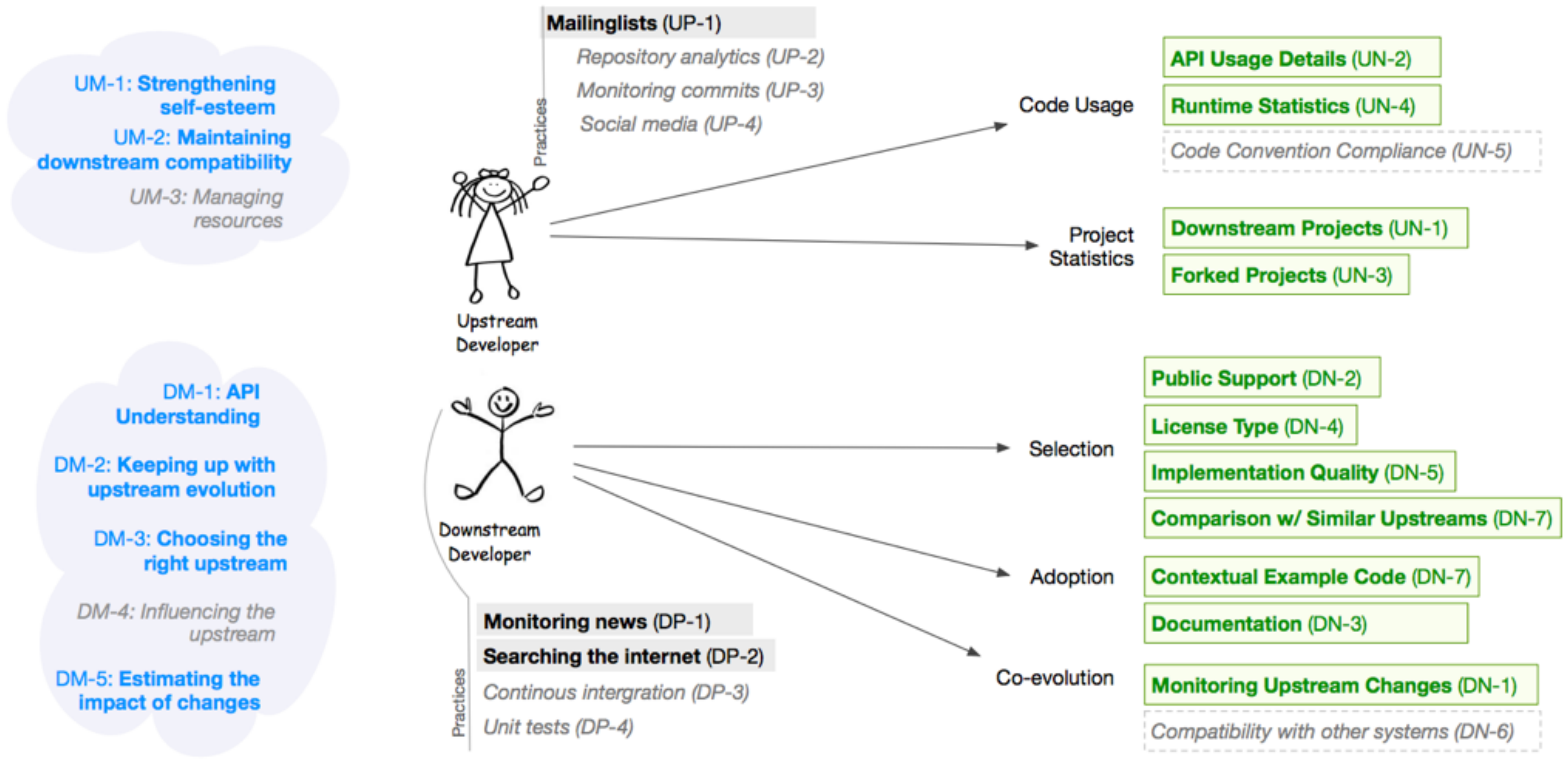
Find all the resources related to a given feature, API, product

Find why a recent change was made

Being notified that a recent change affects an engineer's work

Finding who might be affected by a given change to code/API

Survey of Information Needs in Open Source



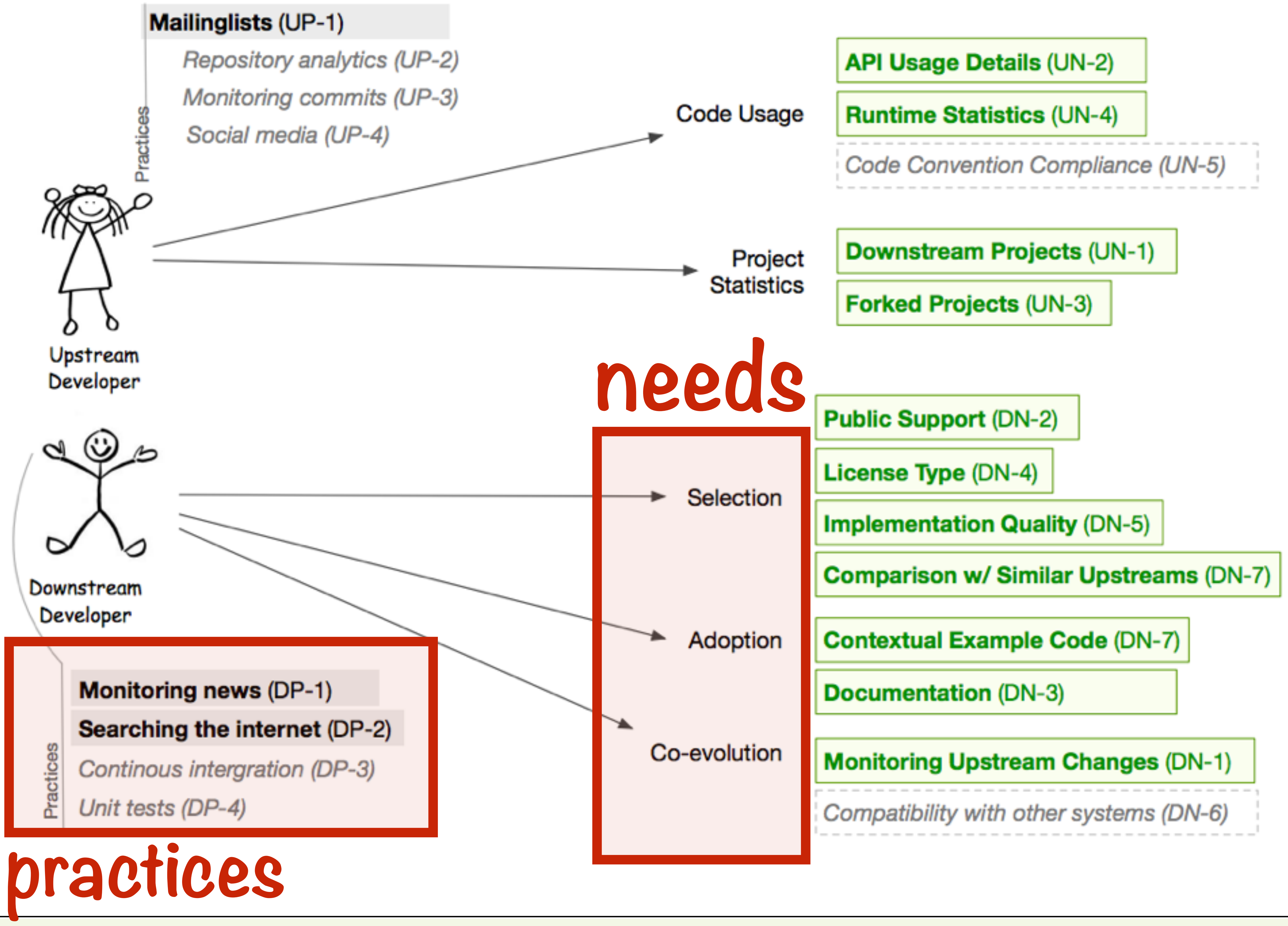
UM-1: Strengthening self-esteem
UM-2: Maintaining downstream compatibility
UM-3: Managing resources
DM-1: API Understanding
DM-2: Keeping up with upstream evolution
DM-3: Choosing the right upstream
DM-4: Influencing the upstream
DM-5: Estimating the impact of changes
Practices
Mailinglists (UP-1)
Repository analytics (UP-2)
Monitoring commits (UP-3)
Social media (UP-4)
Upstream Developer
Downstream Developer
Practices
Monitoring news (DP-1)
Searching the internet (DP-2)
Continous intergration (DP-3)
Unit tests (DP-4)
Code Usage
Project Statistics
Selection
Adoption
Co-evolution
API Usage Details (UN-2)
Runtime Statistics (UN-4)
Code Convention Compliance (UN-5)
Downstream Projects (UN-1)
Forked Projects (UN-3)
Public Support (DN-2)
License Type (DN-4)
Implementation Quality (DN-5)
Comparison w/ Similar Upstreams (DN-7)
Contextual Example Code (DN-7)
Documentation (DN-3)
Monitoring Upstream Changes (DN-1)
Compatibility with other systems (DN-6)

http://scg.unibe.ch/scgbib?query=Haen14a

Open JDK, Processing.js, jQuery, SciPy, NumPy, Pharo, Squeak, Seaside, Drupal, Core-audio, Apache Hadoop, Apache Cassandra, Google WebToolkit, Ubuntu, Soot and Zend Framework

Participants: 75

Technologies


Mailinglists (UP-1)
Repository analytics (UP-2)
Monitoring commits (UP-3)
Social media (UP-4)
Practices
Upstream Developer
Code Usage
Project Statistics
API Usage Details (UN-2)
Runtime Statistics (UN-4)
Code Convention Compliance (UN-5)
Downstream Projects (UN-1)
Forked Projects (UN-3)
needs
Downstream Developer
Selection
Adoption
Co-evolution
Public Support (DN-2)
License Type (DN-4)
Implementation Quality (DN-5)
Comparison w/ Similar Upstreams (DN-7)
Contextual Example Code (DN-7)
Documentation (DN-3)
Monitoring Upstream Changes (DN-1)
Compatibility with other systems (DN-6)
Monitoring news (DP-1)
Searching the internet (DP-2)
Continous intergration (DP-3)
Unit tests (DP-4)
Practices
practices

Upstream

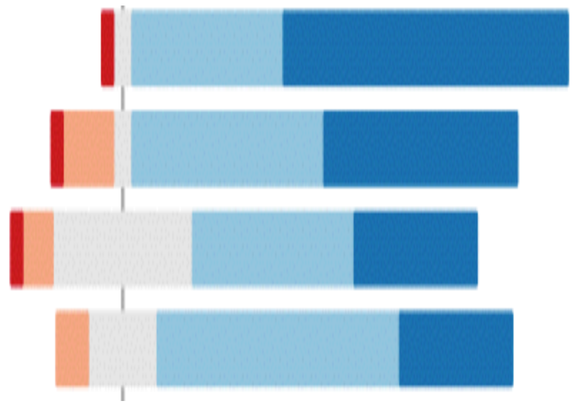
I'm interested in

… the usability of my API

… which API methods are called

… unused methods and functionalities

… how the library is being used



Upstream

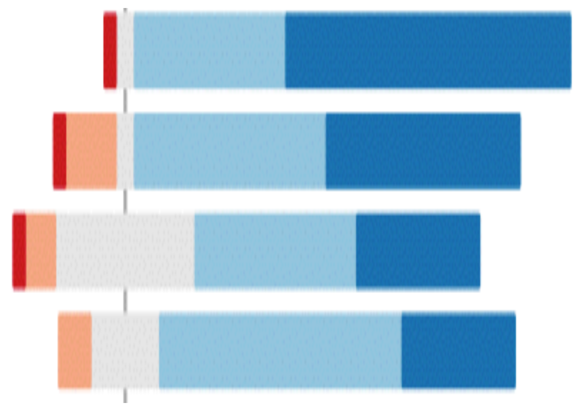
I'm interested in

… the usability of my API

… which API methods are called

… unused methods and functionalities

… how the library is being used



Downstream

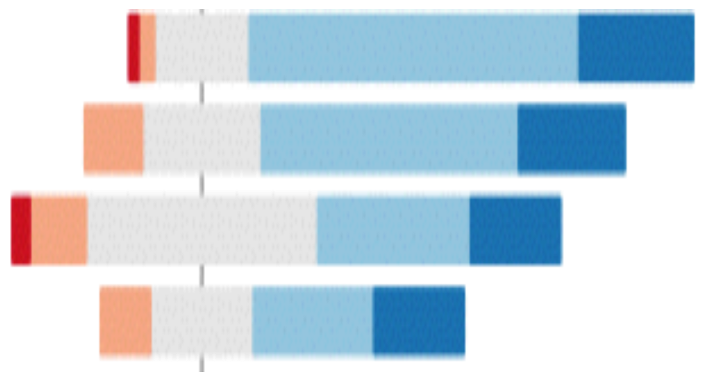
I'm interested in

… **the impact of changes.**

… **the estimated time to adapt to a new version** …

… **notifications about changes**…

*** I keep up to date with my upstream projects** …



Complete list in the paper…

What you should know!

> What is an ecosystem and why talk about it
> Opportunities associated with ecosystems
> Challenges that appear in live ecosystems

Can you answer these questions?

- What is *dependency hell*? What are some solutions?
- How would you mine library usage from the ecosystem?
- How would you approach detecting clones in a large ecosystem?
- What are the challenges for a developer working in an ecosystem?
- What are the benefits of software ecosystems?

Further Reading

Mandatory Reading

> **The Cathedral and the Bazaar,** Erik S. Raymond

Optional

> **The Law of Leaky Abstractions,** Joel Spolsky

> **Mining Trends in Library Usage**, Mileva et al. 2009

> **Codebook: Discovering and Exploiting Relationships in Software Repositories,** Begel et al. 2010