Extracting the Essence of Software Systems' Architectures through Unstructured-Data Mining

Nenad Medvidović

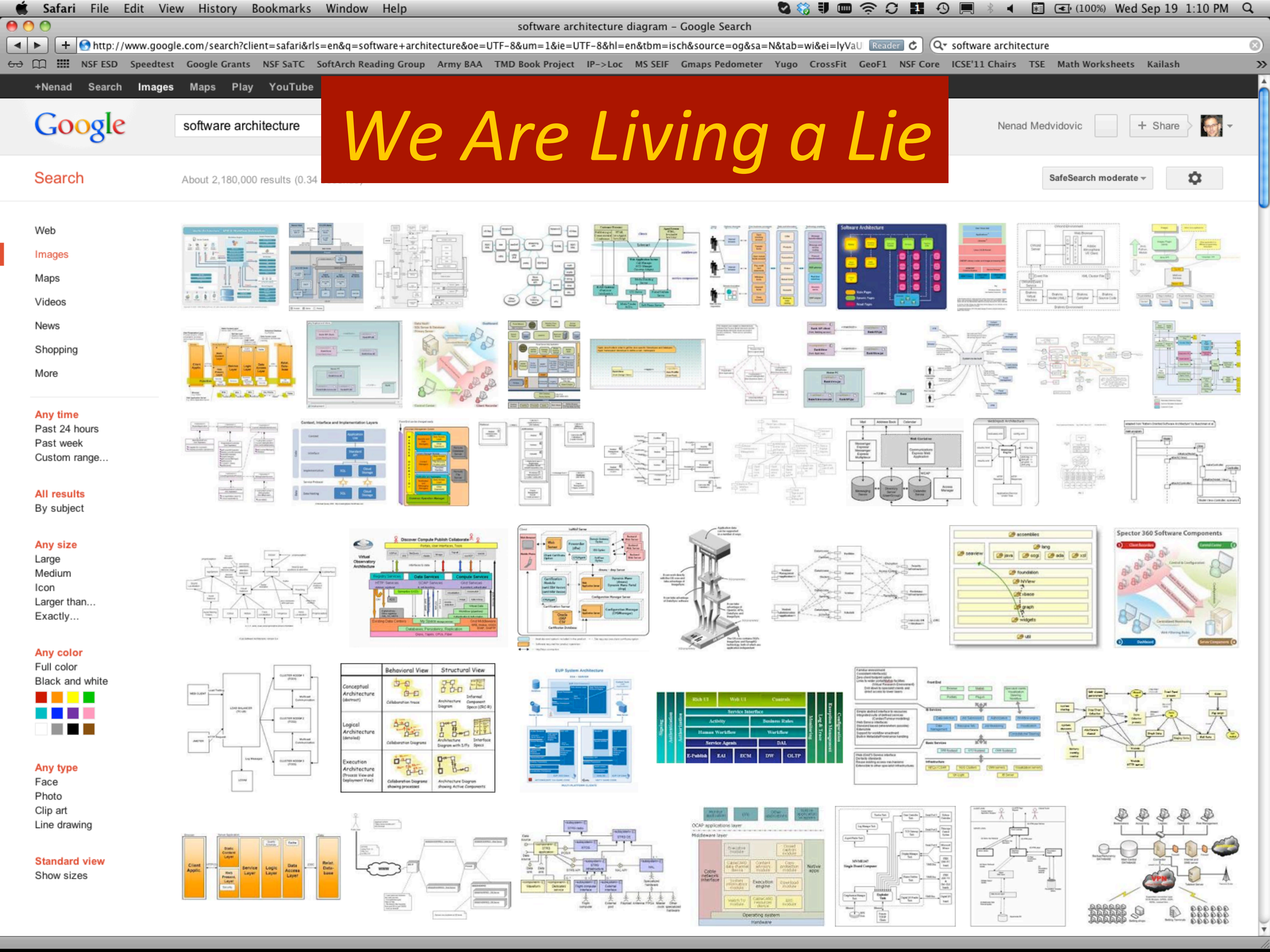
University of Southern California

Los Angeles, CA, USA

neno@usc.edu

http://csse.usc.edu/~neno/

We Are Living a Lie



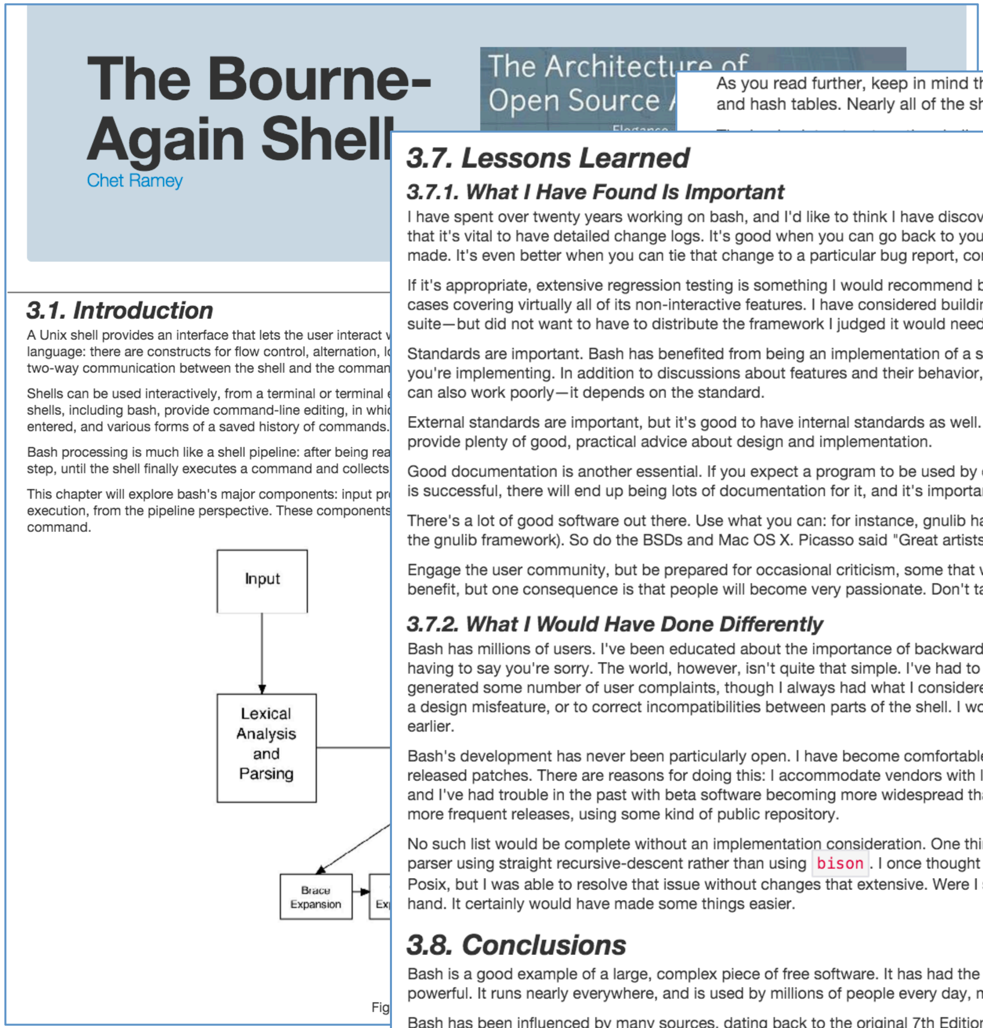
software architecture diagram - Google Search

http://www.google.com/search?client=safari&rls=en&q=software+architecture&oe=UTF-8&um=1&ie=UTF-8&hl=en&tbm=isch&source=og&sa=N&tab=wi&ei=lyVaU

software architecture

About 2,180,000 results (0.34

Real Architecture Is… Unstructured



The Bourne-Again Shell

Chet Ramey

The Architecture of Open Source

3.1. Introduction

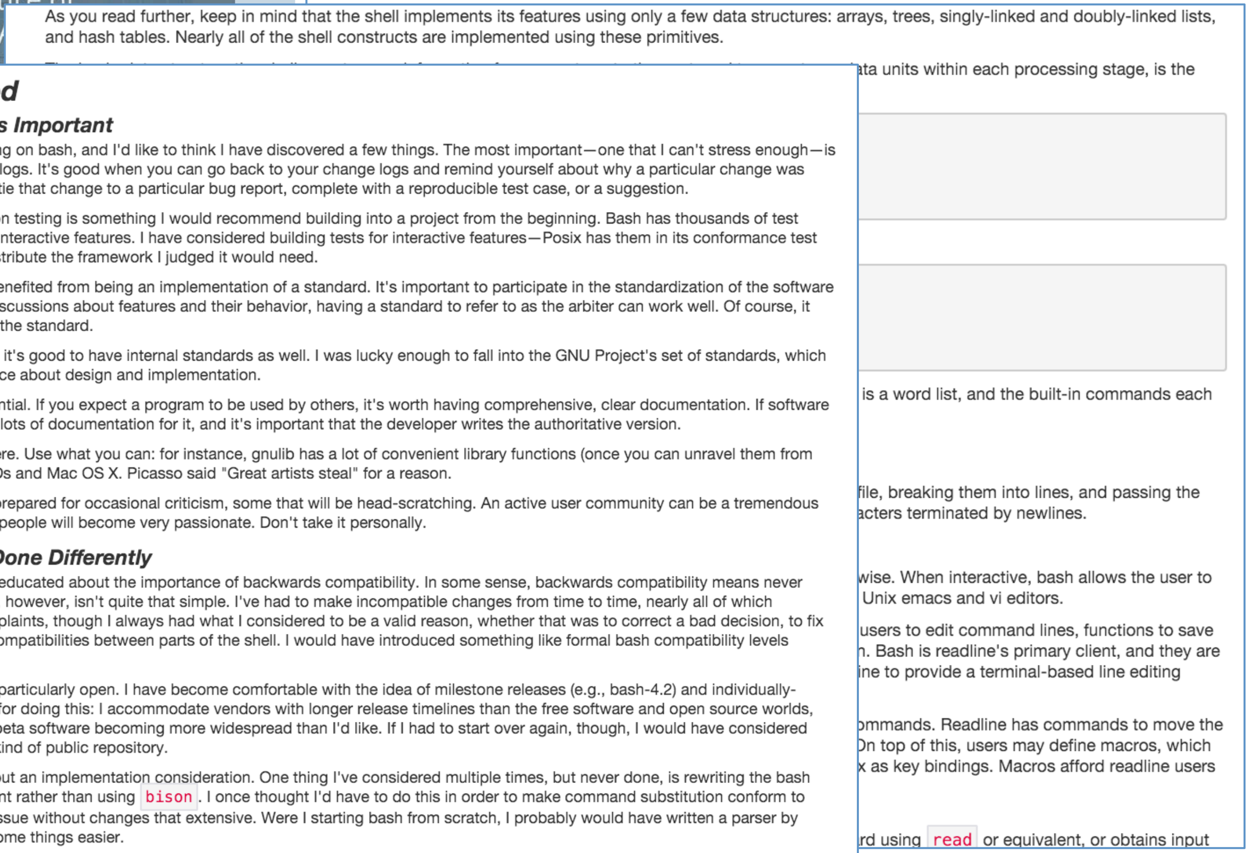
A Unix shell provides an interface that lets the user interact
language: there are constructs for flow control, alternation, l
two-way communication between the shell and the comman

Shells can be used interactively, from a terminal or terminal
shells, including bash, provide command-line editing, in whic
entered, and various forms of a saved history of commands.

Bash processing is much like a shell pipeline: after being rea
step, until the shell finally executes a command and collects

This chapter will explore bash's major components: input pr
execution, from the pipeline perspective. These components
command.

Fig



As you read further, keep in mind that the shell implements its features using only a few data structures: arrays, trees, singly-linked and doubly-linked lists, and hash tables. Nearly all of the shell constructs are implemented using these primitives.

ata units within each processing stage, is the

is a word list, and the built-in commands each

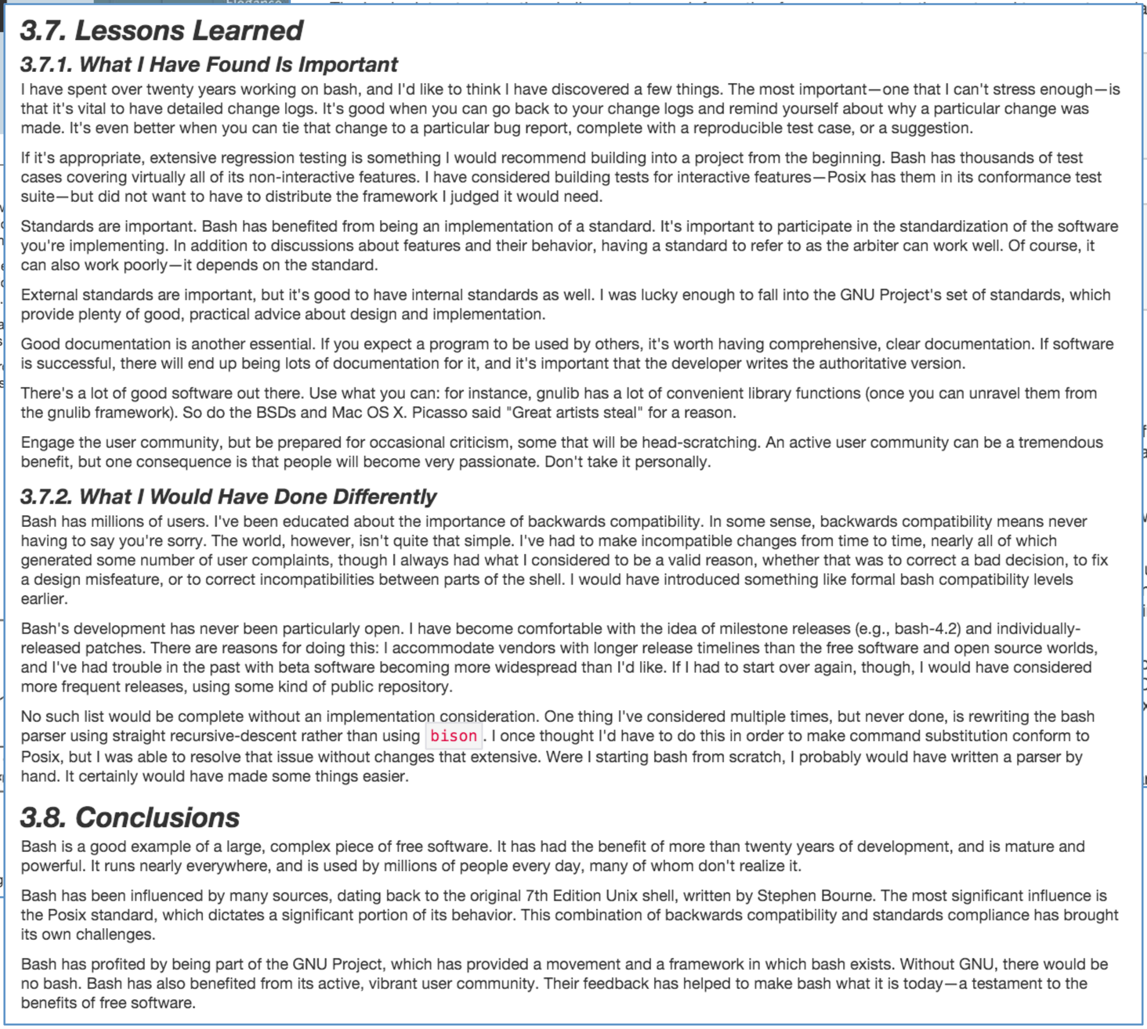
file, breaking them into lines, and passing the
acters terminated by newlines.

wise. When interactive, bash allows the user to
Unix emacs and vi editors.

users to edit command lines, functions to save
n. Bash is readline's primary client, and they are
ine to provide a terminal-based line editing

ommands. Readline has commands to move the
On top of this, users may define macros, which
x as key bindings. Macros afford readline users

rd using `read` or equivalent, or obtains input



3.7. Lessons Learned

3.7.1. What I Have Found Is Important

I have spent over twenty years working on bash, and I'd like to think I have discovered a few things. The most important—one that I can't stress enough—is that it's vital to have detailed change logs. It's good when you can go back to your change logs and remind yourself about why a particular change was made. It's even better when you can tie that change to a particular bug report, complete with a reproducible test case, or a suggestion.

If it's appropriate, extensive regression testing is something I would recommend building into a project from the beginning. Bash has thousands of test cases covering virtually all of its non-interactive features. I have considered building tests for interactive features—Posix has them in its conformance test suite—but did not want to have to distribute the framework I judged it would need.

Standards are important. Bash has benefited from being an implementation of a standard. It's important to participate in the standardization of the software you're implementing. In addition to discussions about features and their behavior, having a standard to refer to as the arbiter can work well. Of course, it can also work poorly—it depends on the standard.

External standards are important, but it's good to have internal standards as well. I was lucky enough to fall into the GNU Project's set of standards, which provide plenty of good, practical advice about design and implementation.

Good documentation is another essential. If you expect a program to be used by others, it's worth having comprehensive, clear documentation. If software is successful, there will end up being lots of documentation for it, and it's important that the developer writes the authoritative version.

There's a lot of good software out there. Use what you can: for instance, gnulib has a lot of convenient library functions (once you can unravel them from the gnulib framework). So do the BSDs and Mac OS X. Picasso said "Great artists steal" for a reason.

Engage the user community, but be prepared for occasional criticism, some that will be head-scratching. An active user community can be a tremendous benefit, but one consequence is that people will become very passionate. Don't take it personally.

3.7.2. What I Would Have Done Differently

Bash has millions of users. I've been educated about the importance of backwards compatibility. In some sense, backwards compatibility means never having to say you're sorry. The world, however, isn't quite that simple. I've had to make incompatible changes from time to time, nearly all of which generated some number of user complaints, though I always had what I considered to be a valid reason, whether that was to correct a bad decision, to fix a design misfeature, or to correct incompatibilities between parts of the shell. I would have introduced something like formal bash compatibility levels earlier.

Bash's development has never been particularly open. I have become comfortable with the idea of milestone releases (e.g., bash-4.2) and individually-released patches. There are reasons for doing this: I accommodate vendors with longer release timelines than the free software and open source worlds, and I've had trouble in the past with beta software becoming more widespread than I'd like. If I had to start over again, though, I would have considered more frequent releases, using some kind of public repository.

No such list would be complete without an implementation consideration. One thing I've considered multiple times, but never done, is rewriting the bash parser using straight recursive-descent rather than using `bison`. I once thought I'd have to do this in order to make command substitution conform to Posix, but I was able to resolve that issue without changes that extensive. Were I starting bash from scratch, I probably would have written a parser by hand. It certainly would have made some things easier.

3.8. Conclusions

Bash is a good example of a large, complex piece of free software. It has had the benefit of more than twenty years of development, and is mature and powerful. It runs nearly everywhere, and is used by millions of people every day, many of whom don't realize it.

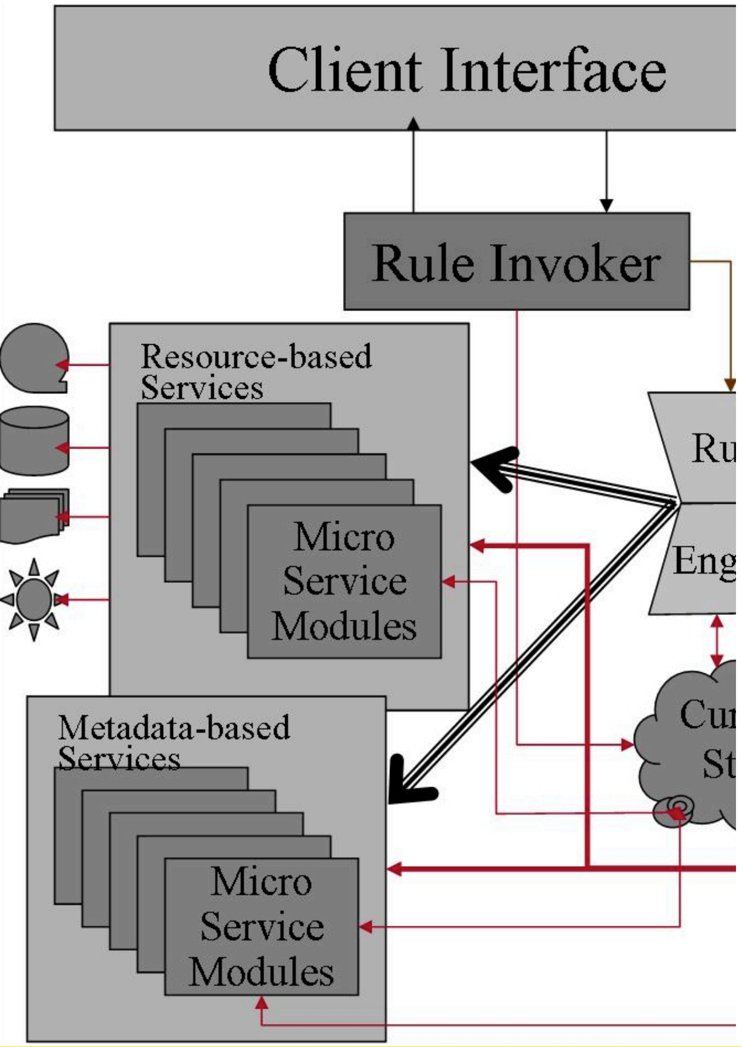
Bash has been influenced by many sources, dating back to the original 7th Edition Unix shell, written by Stephen Bourne. The most significant influence is the Posix standard, which dictates a significant portion of its behavior. This combination of backwards compatibility and standards compliance has brought its own challenges.

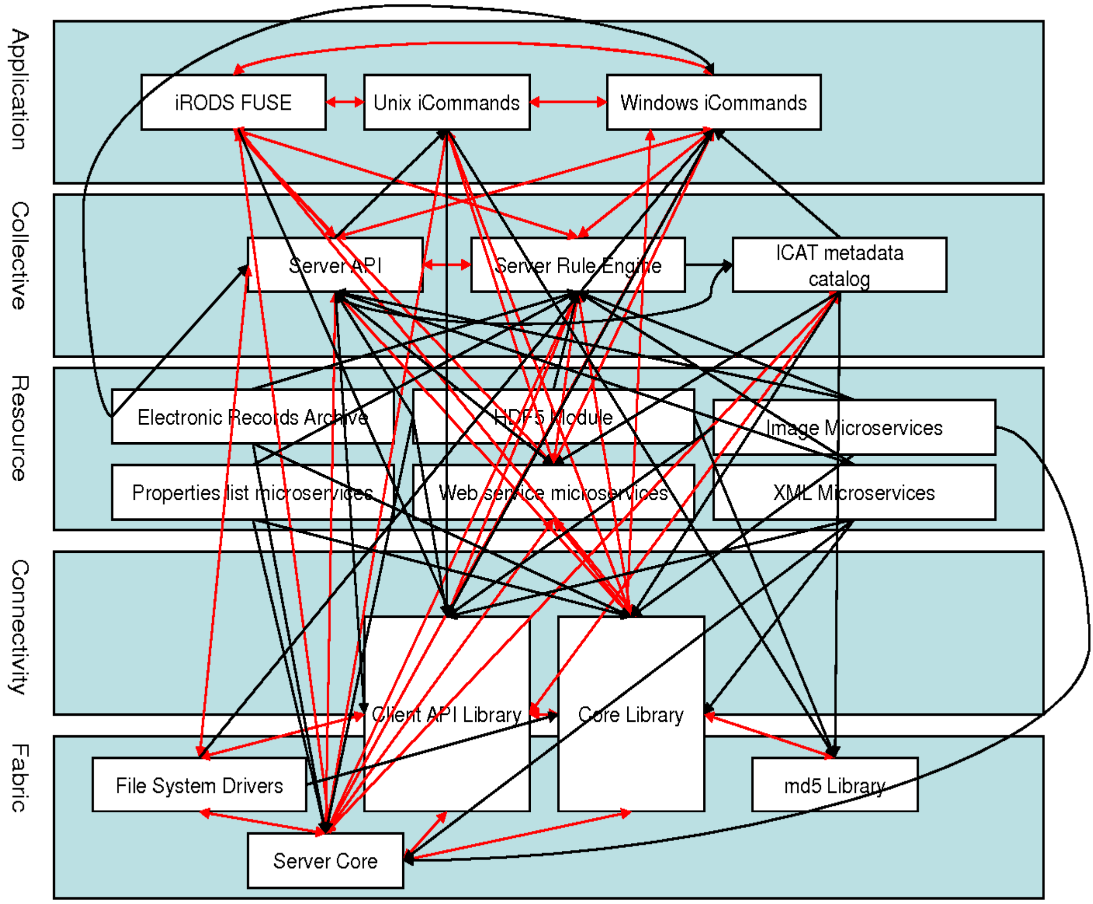
Bash has profited by being part of the GNU Project, which has provided a movement and a framework in which bash exists. Without GNU, there would be no bash. Bash has also benefited from its active, vibrant user community. Their feedback has helped to make bash what it is today—a testament to the benefits of free software.

How Many Systems Start off

iRODS – Prescriptive Architecture

...and End up





iRODS – Descriptive Architecture

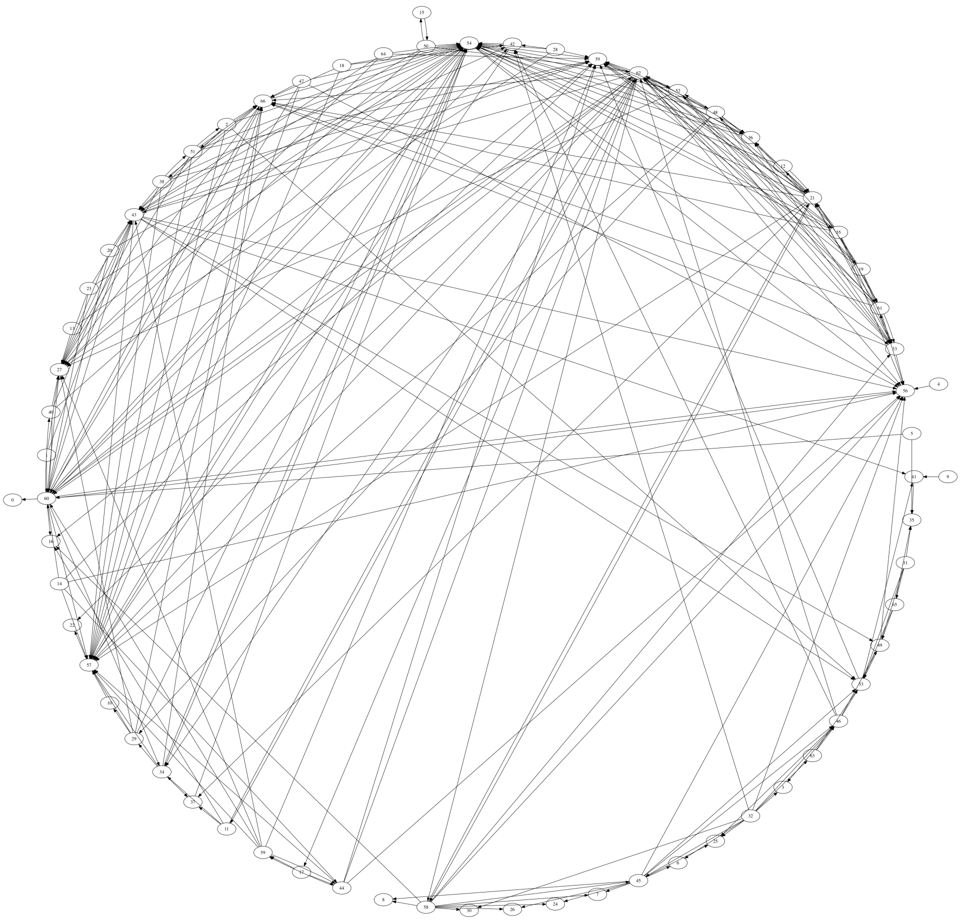
What Happened?

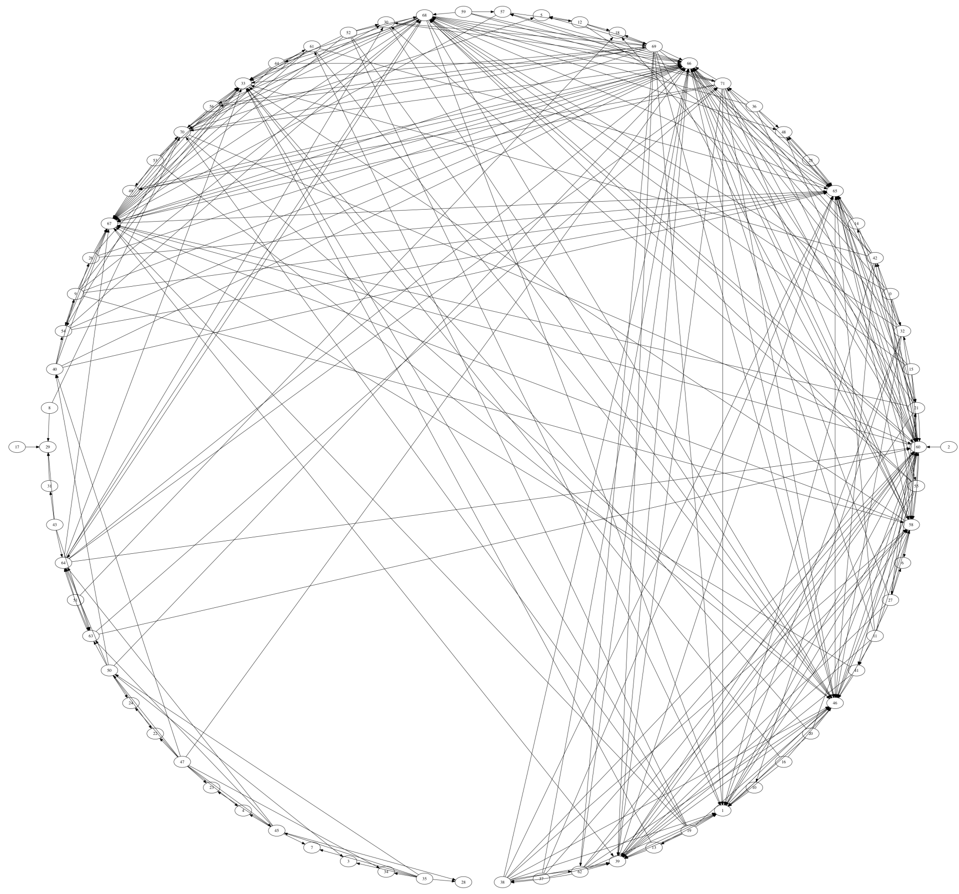
- Software *decay*
  - *Drift* – introduction of design decisions into a system that are not encompassed or implied by its architectural design
  - *Erosion* – introduction of design decisions into a system that violate its architectural design

At What Point Does Change Become Decay?

Apache Chukwa 0.3.0

Apache Chukwa 0.4.0





USC

School of Engineering

University of Southern California

Can We "Smell" Decay?

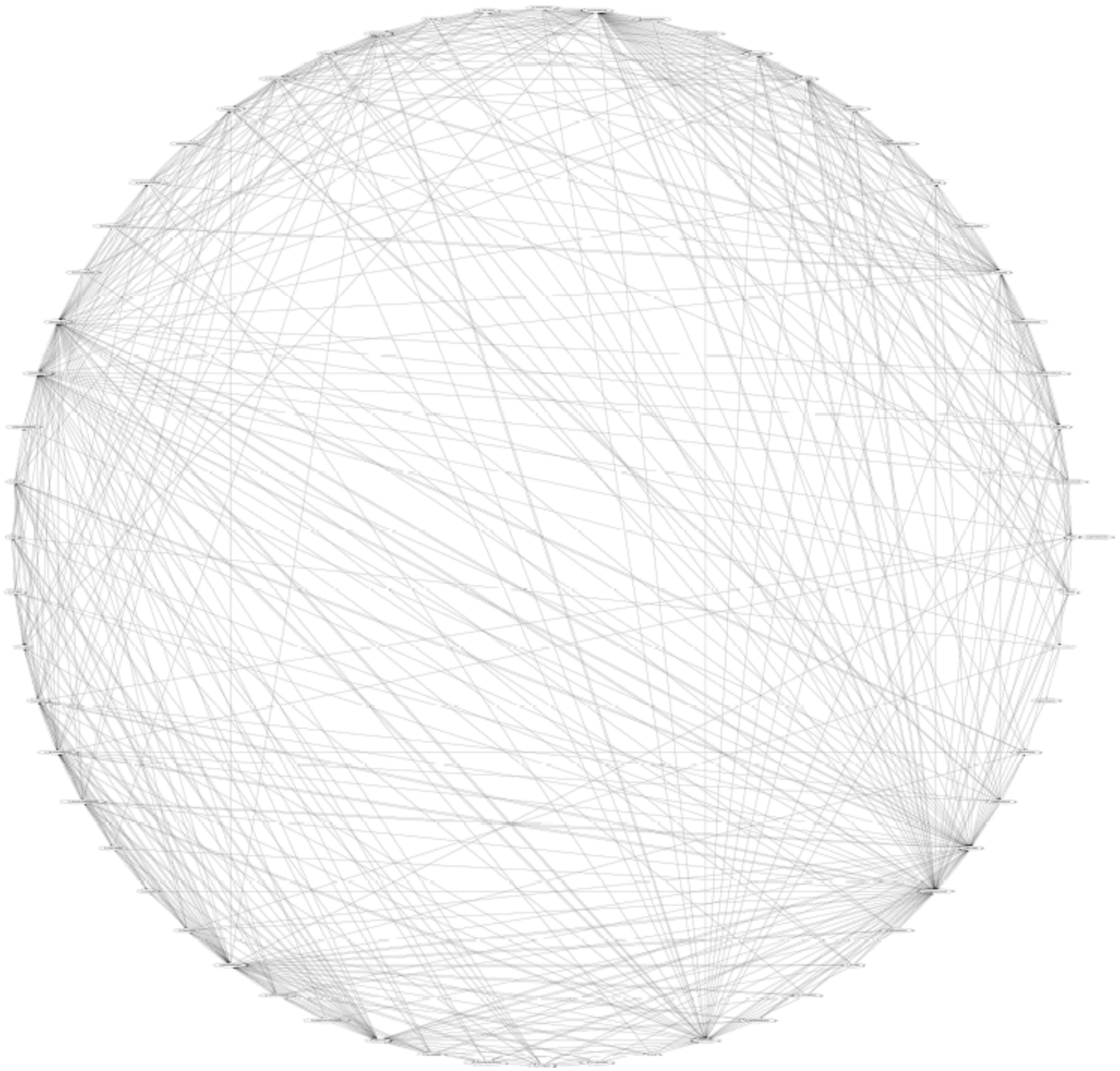
- Yes, both in the design and code
- Software smell
  - Commonly made design or implementation decision
  - Negatively impacts your system's lifecycle properties
  - It is not a bug – it doesn't break your system
- Our goal is to discover architectural design smells automatically
- Inspired by
  - *Refactoring: Improving the Design of Existing Code*
    by Martin Fowler

A Catalogue of Architectural Smells

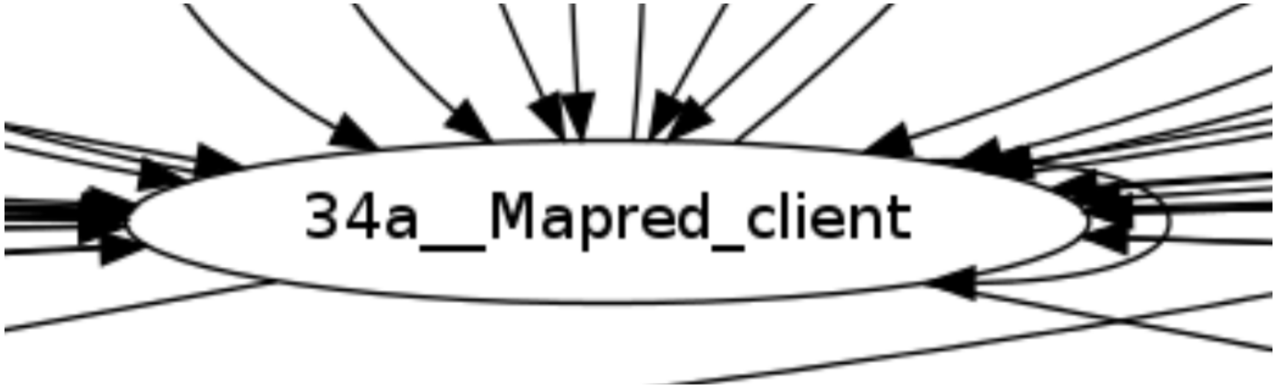
- Brick Concern Overload
- Brick Use Overload
- Brick Dependency Cycle
- Unused Interface
- Ambiguous Interface
- Duplicate Component Functionality
- Scattered Functionality
- Component Envy
- Connector Envy
- Connector Chain
- Extraneous Adjacent Connector
- …

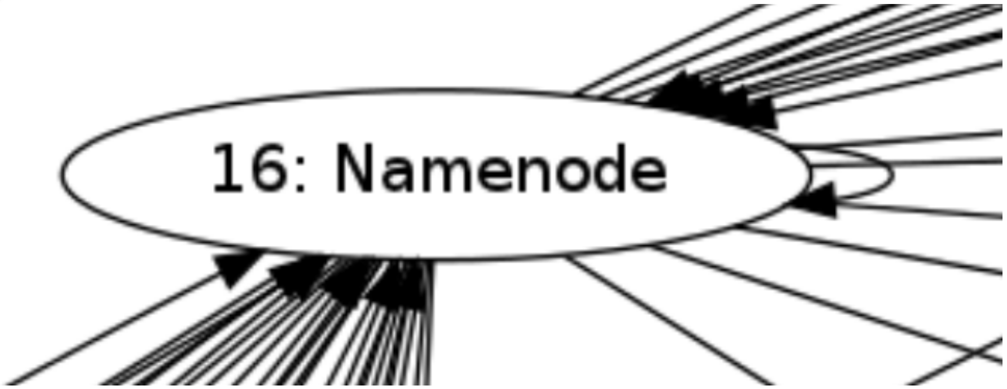
Example: Hadoop's Recovered

Hadoop – Dependency Cycle

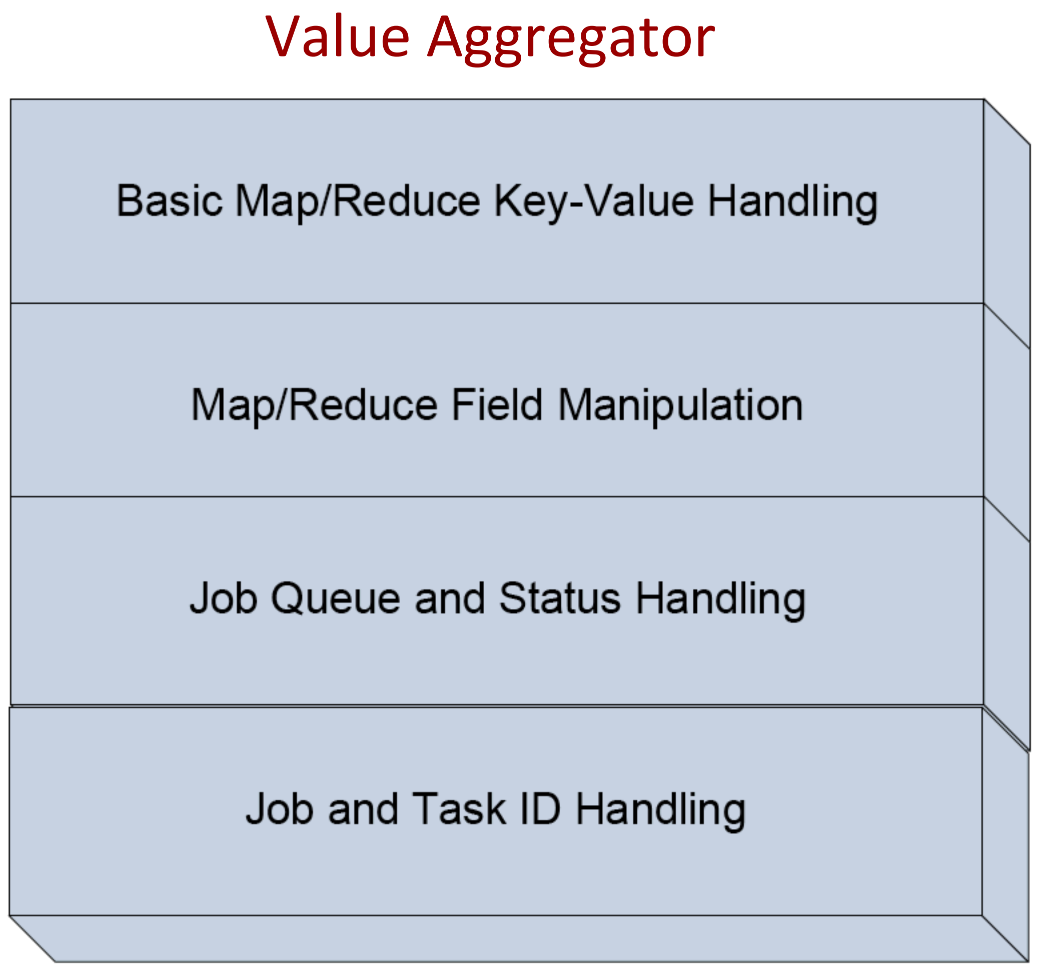


Hadoop – Component Use Overload



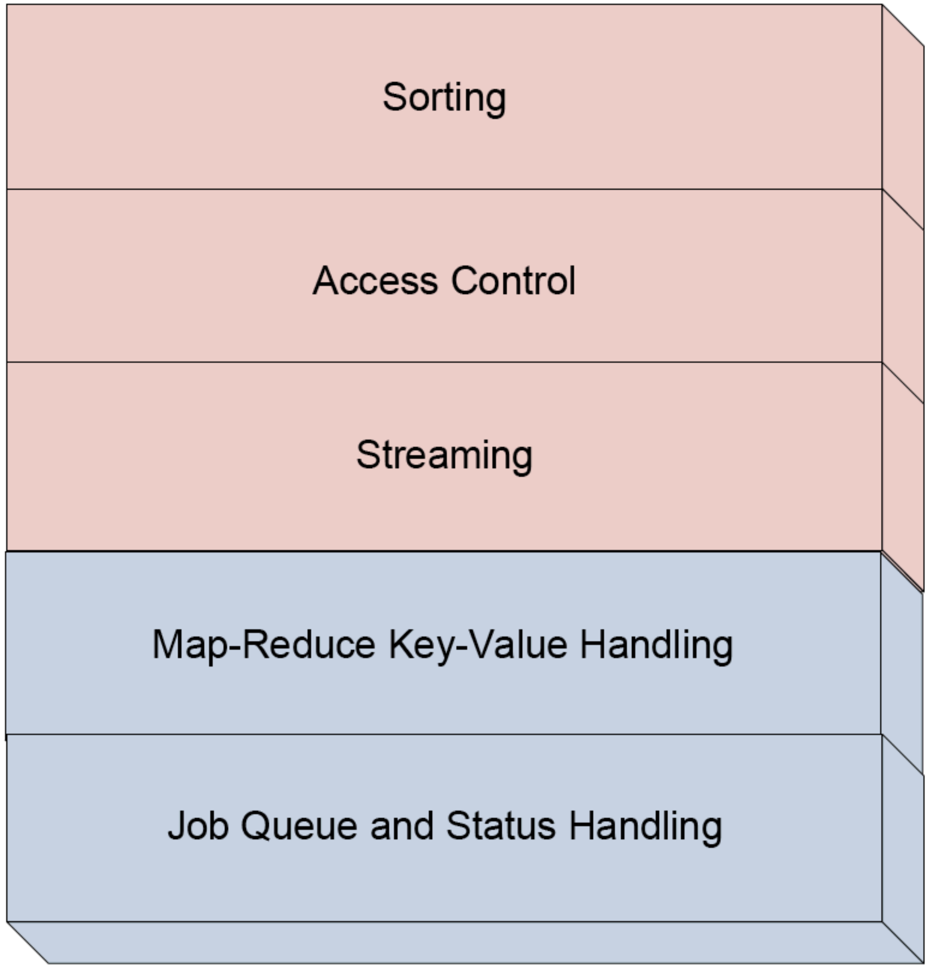


Hadoop – Concern Overload


Value Aggregator
Basic Map/Reduce Key-Value Handling
Map/Reduce Field Manipulation
Job Queue and Status Handling
Job and Task ID Handling

Hadoop – * Envy

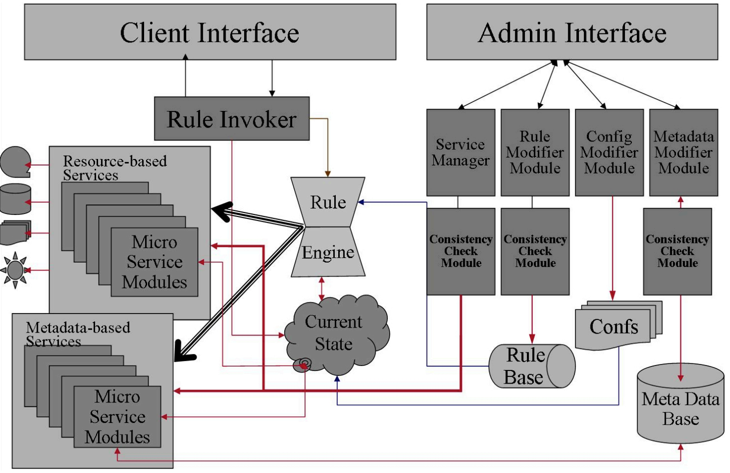
InterDataNode Protocol

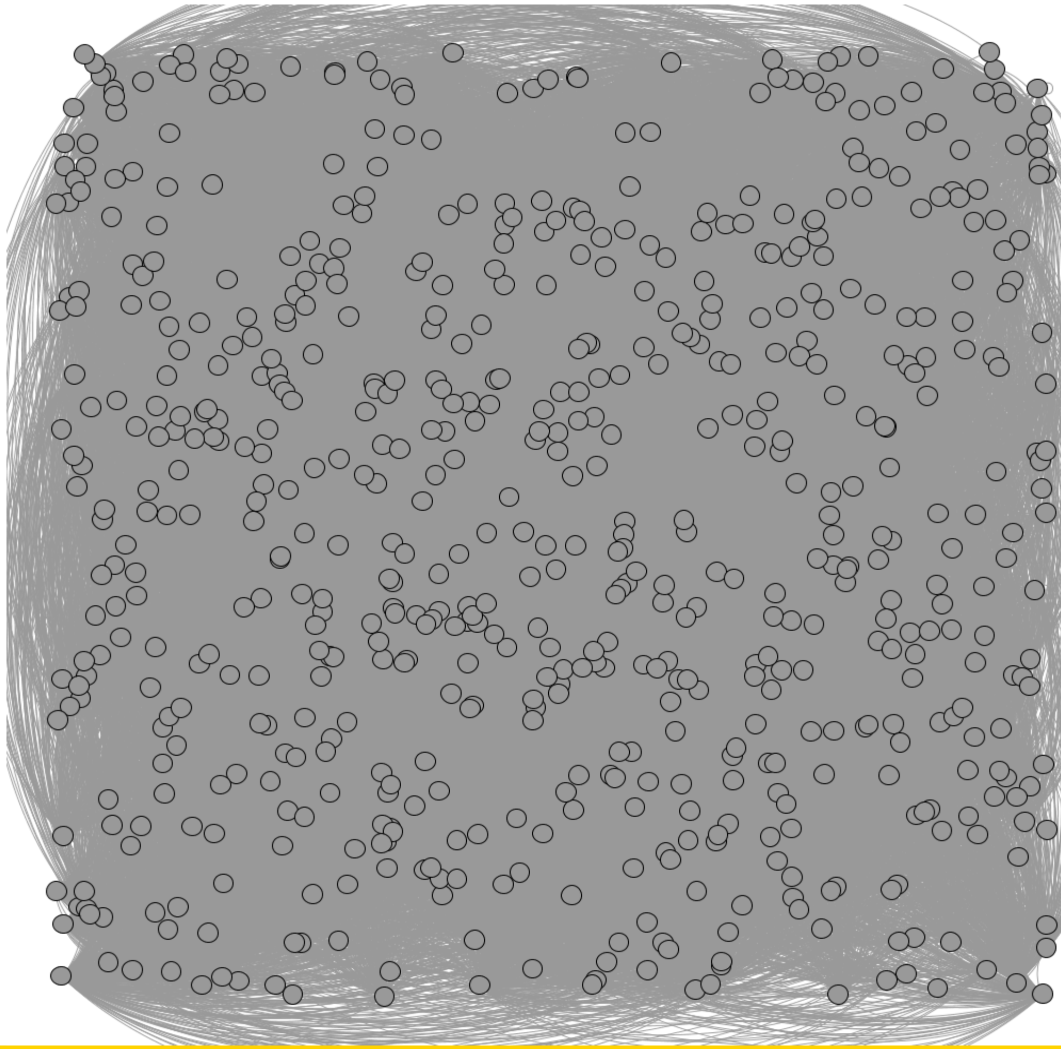


How Many Systems Start off

iRODS – Prescriptive Architecture

…and End up





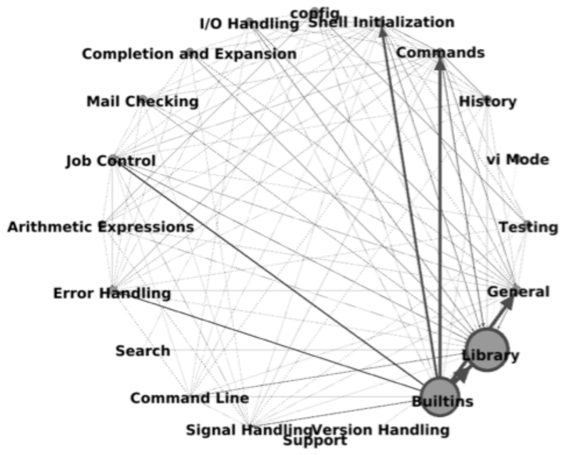
What Can Be Done?

- *Architecture recovery*
  - The process of determining a system's architecture from its implementation-level artifacts and many other information sources
  - Source code, executable files, Java .class files, …
- Difficult in practice
  - Size of code bases
  - Irrelevant details
  - Misleading details
  - Missing information
  - Lots and lots of unstructured data

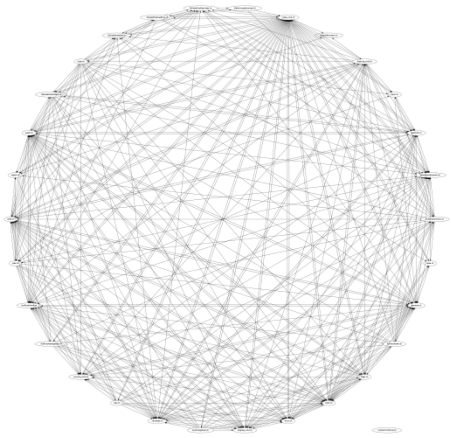
Automated Solutions Are Available

- ACDC – Algorithm for Comprehension-Driven Clustering
  - Structural pattern-based clustering
- ARC – Architecture Recovery Using Concerns
  - Concern-based hierarchical clustering based on similarity measure
- Bunch-NAHC & Bunch-SAHC
  - Hill-climbing algorithm for maximizing *Modularization Quality*
- LIMBO – scaLable InforMation BOttleneck
  - Probabilistic hierarchical clustering
- WCA-UE & WCA-UENM – Weigted Combined Algorithm
  - Dependency-based hierarchical clustering
- ZBR – Zone-Based Recovery
  - Hierarchical clustering based on textual information

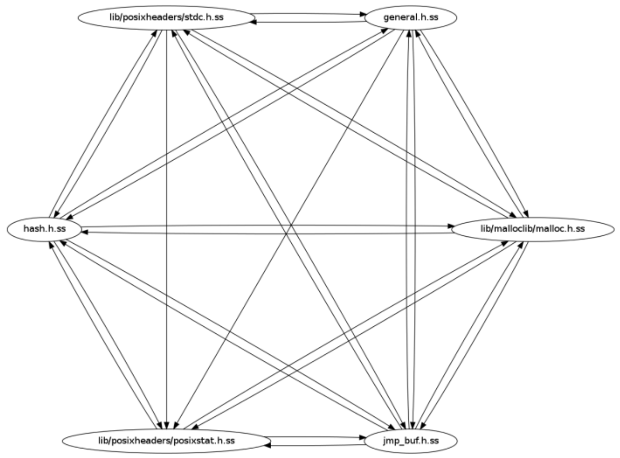
Different Architectural Views of Bash



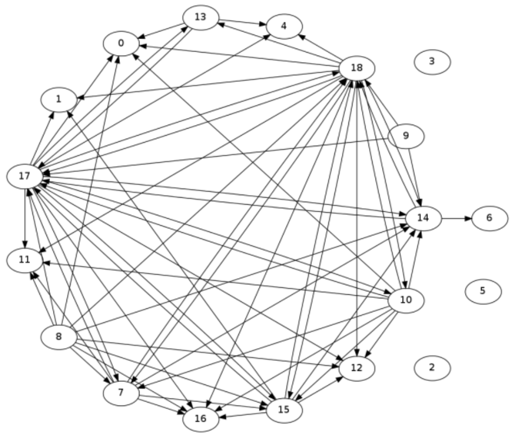
Manual



ACDC


lib/posixheaders/stdc.h.ss
general.h.ss
hash.h.ss
lib/malloclib/malloc.h.ss
lib/posixheaders/posixstat.h.ss
jmp_buf.h.ss

Bunch

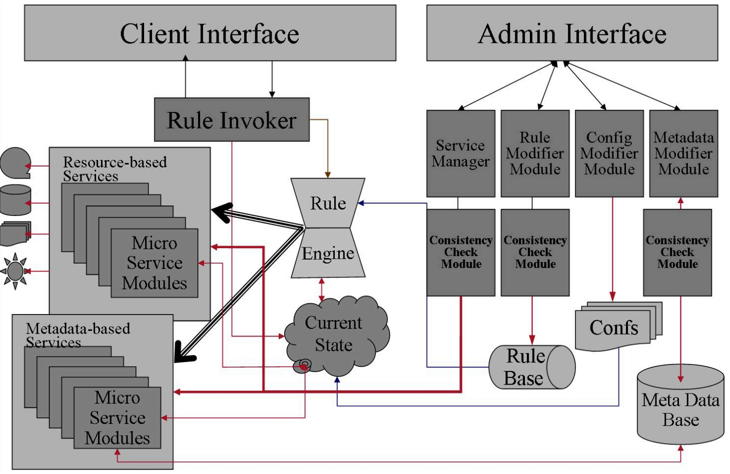


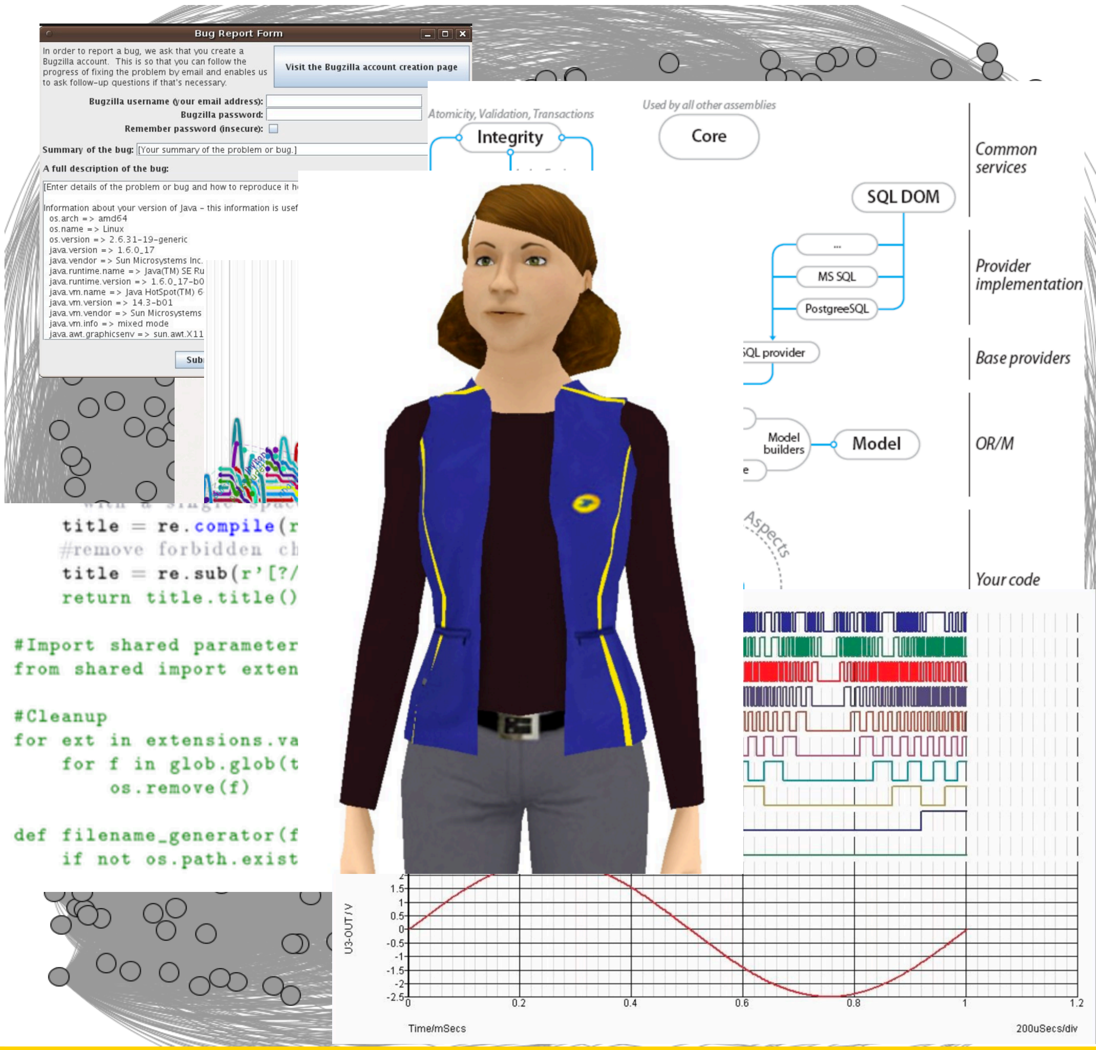
ZBR

How Many Systems Start off

iRODS – Prescriptive Architecture

…and End up




Bug Report Form

In order to report a bug, we ask that you create a Bugzilla account. This is so that you can follow the progress of fixing the problem by email and enables us to ask follow-up questions if that's necessary.

Visit the Bugzilla account creation page

Bugzilla username (your email address):
Bugzilla password:
Remember password (insecure):

Summary of the bug: [Your summary of the problem or bug.]

A full description of the bug:

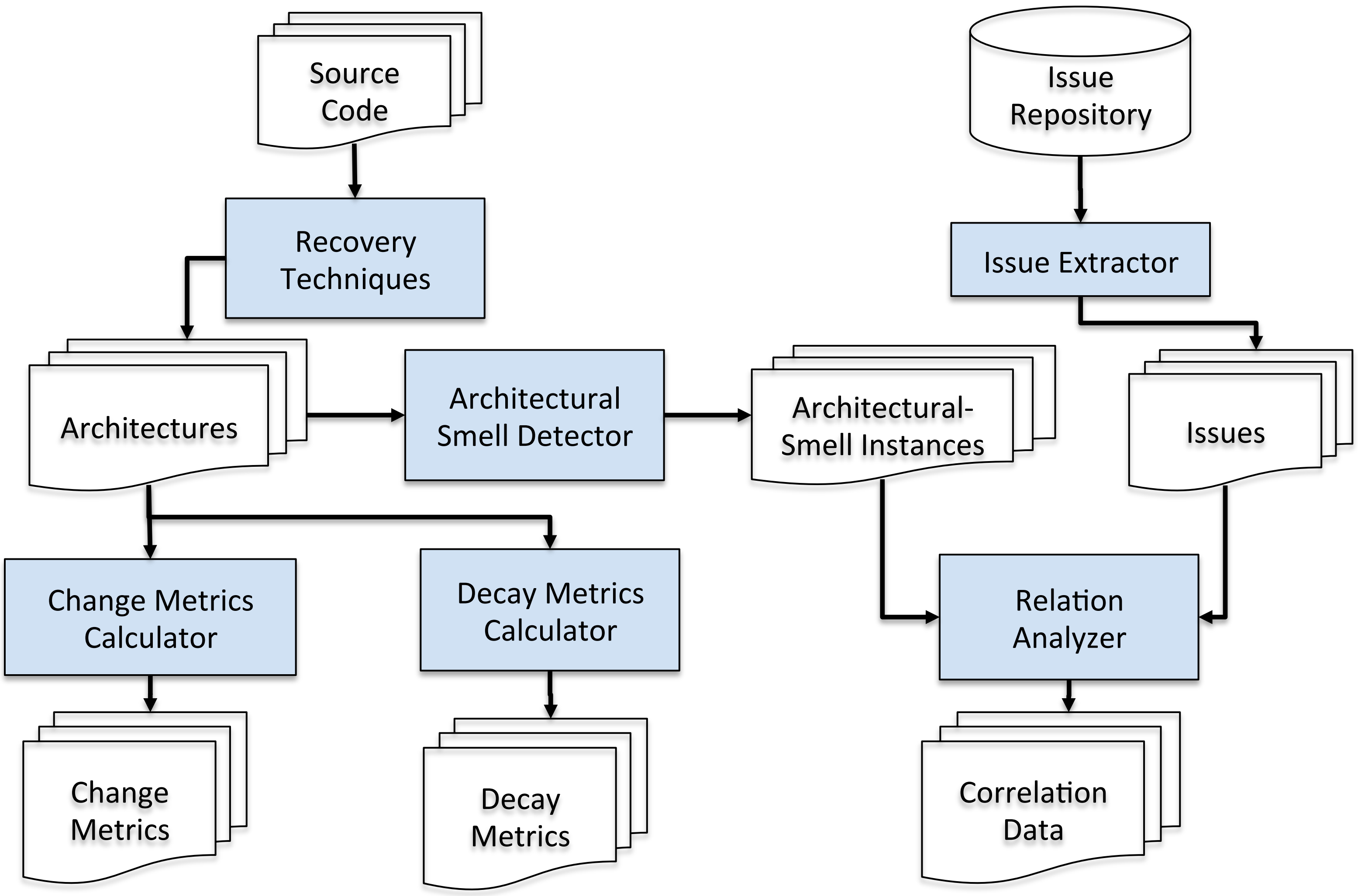
```
[Enter details of the problem or bug and how to reproduce it h
Information about your version of Java - this information is usef
 os.arch => amd64
 os.name => Linux
 os.version => 2.6.31-19-generic
 java.version => 1.6.0_17
 java.vendor => Sun Microsystems Inc.
 java.runtime.name => Java(TM) SE Ru
 java.runtime.version => 1.6.0_17-b0
 java.vm.name => Java HotSpot(TM) 6
 java.vm.version => 14.3-b01
 java.vm.vendor => Sun Microsystems
 java.vm.info => mixed mode
 java.awt.graphicsenv => sun.awt.X11
```

```
    title = re.compile(r
    #remove forbidden ch
    title = re.sub(r'[?/
    return title.title()

#Import shared parameter
from shared import exten

#Cleanup
for ext in extensions.va
    for f in glob.glob(t
        os.remove(f)

def filename_generator(f
    if not os.path.exist
```

Software Architecture as a “Big Data” problem

What We Can Do

ARCADE 1.0 – Architecture Recovery, Change, and Decay Evaluator



Empirical Study of Change and Decay

CHANGE causes DECAY

1. In what ways do architectures change?

2. When and how do architectures decay?

3. What is the relationship between architectural smells and implementation issues?

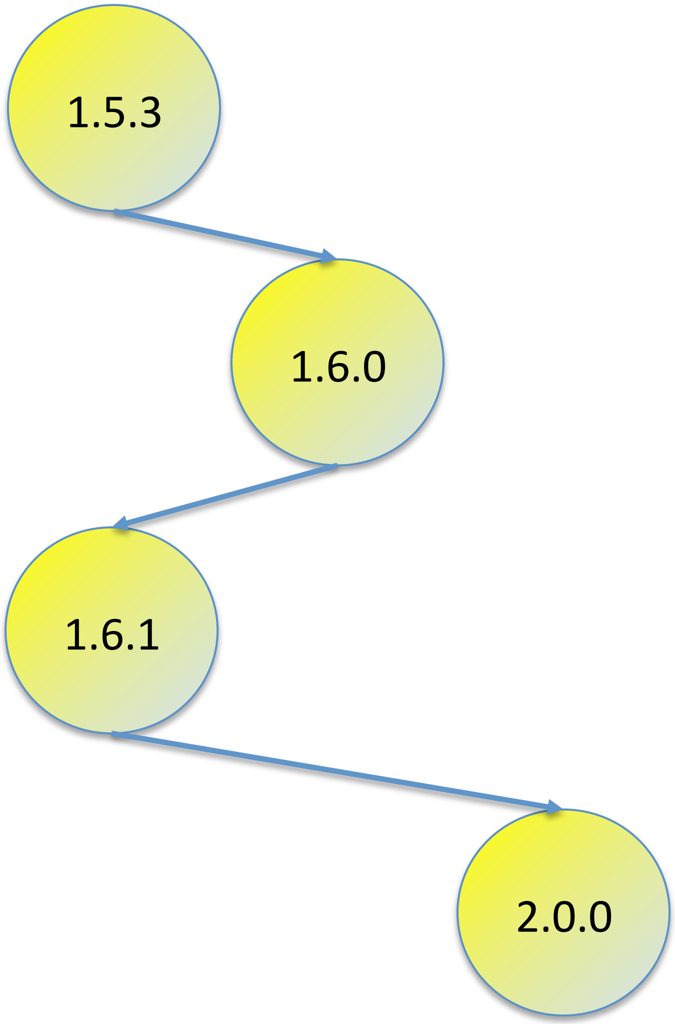
Subject Systems

System	Application Domain	Versions	Time	MSLOC
ActiveMQ	Message Broker	20	8/04-12/05	3.4
Cassandra	Distributed DBMS	127	9/09-9/13	22.0
Chukwa	Data Monitoring	7	5/09-2/14	2.2
Hadoop	Data Processing	63	4/06-8/13	30.0
Ivy	Dependency Manager	20	12/07-2/14	0.4
JackRabbit	Content Repository	97	8/04-2/14	34.0
Jena	Semantic Web Framework	7	6/12-9/13	2.7
JSPWiki	Wiki Engine	54	10/07-3/14	1.2
Log4j	Logging	41	01/01-06/14	2.4
Lucene	Search Engine	21	12/10-1/14	5.1
Mina	Network Framework	40	11/06-11/12	2.3
PDFBox	PDF Library	17	2/08-3/14	2.7
Struts	Web Apps	36	3/00-2/14	6.7
Xerces	XML Library	22	3/03-11/09	2.3

…and many more

A Few Background Bits

- Versioning Scheme
  - major.minor.patch release
- Change metrics
  - MojoFM
  - a2a
  - c2c
- Decay metrics
  - # structural dependencies
  - Change proneness
  - Coupling and cohesion
  - Smell density and coverage



Recovery Techniques Used

- **PKG** – package structure recovery
- **ACDC**[*] – algorithm for comprehension-driven clustering
- **ARC**[**] – architecture recovery using concerns

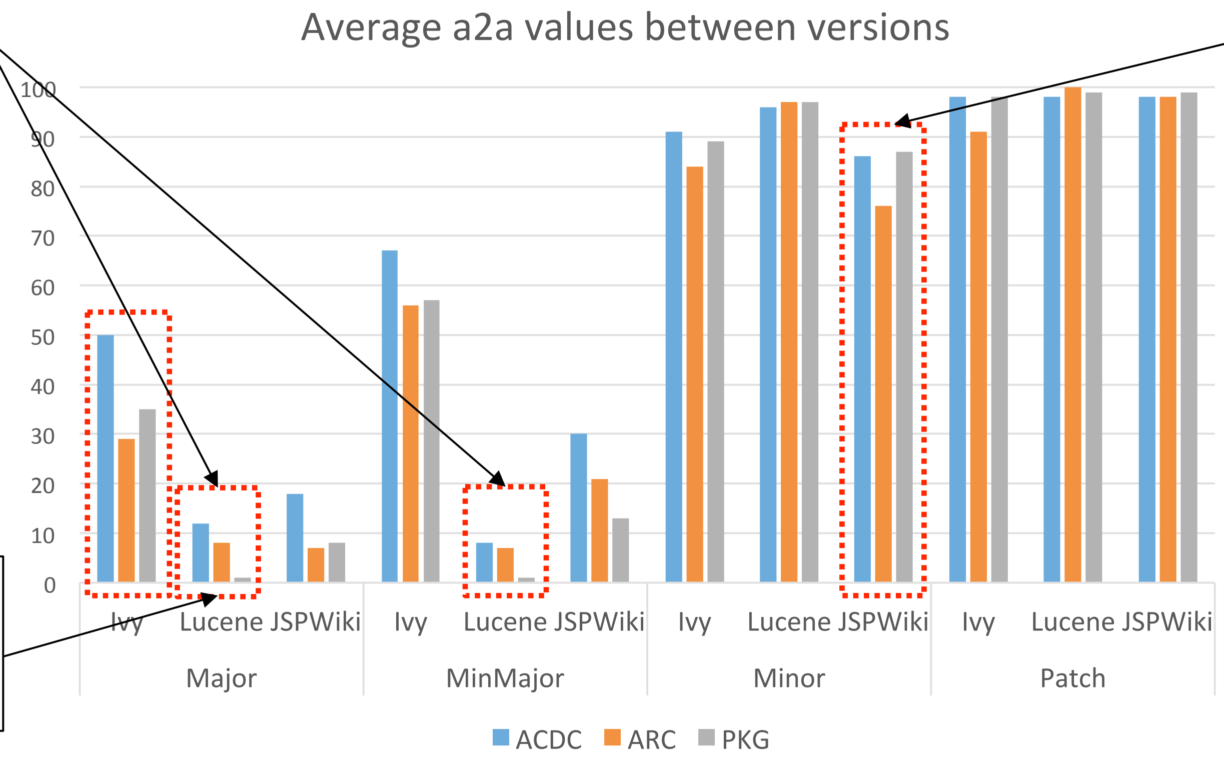
* V. Tzerpos et al., ***ACDC: an algorithm for comprehension-driven clustering***, In Working Conference on Reverse Engineering (WCRE), 2000

** J. Garcia et al., ***Enhancing architectural recovery using concerns***, In International Conference on Automated Software Engineering (ASE), 2011

How Architectures Change

“Reversed” architecture changes

On average, architecture changes range from 15-25%



Changes differ between different views

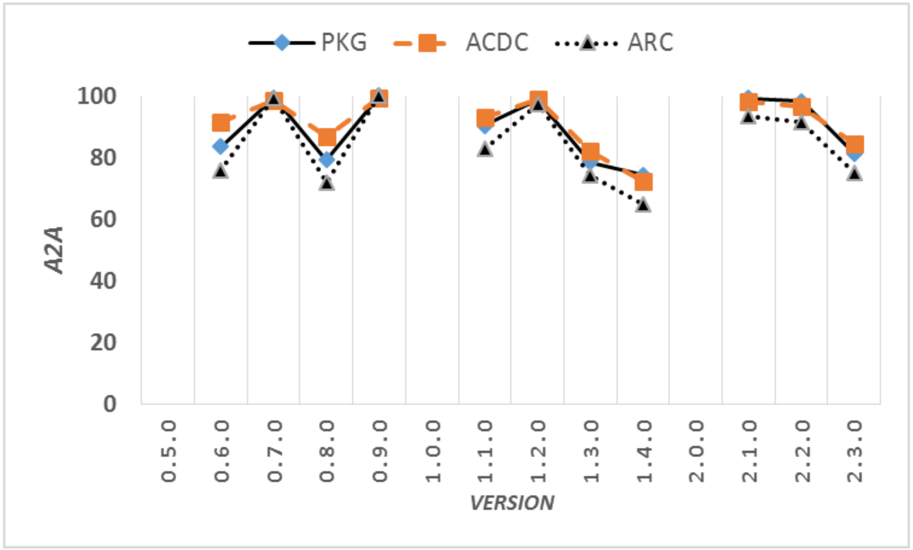
Value unit is percentage
Lower numbers mean more change

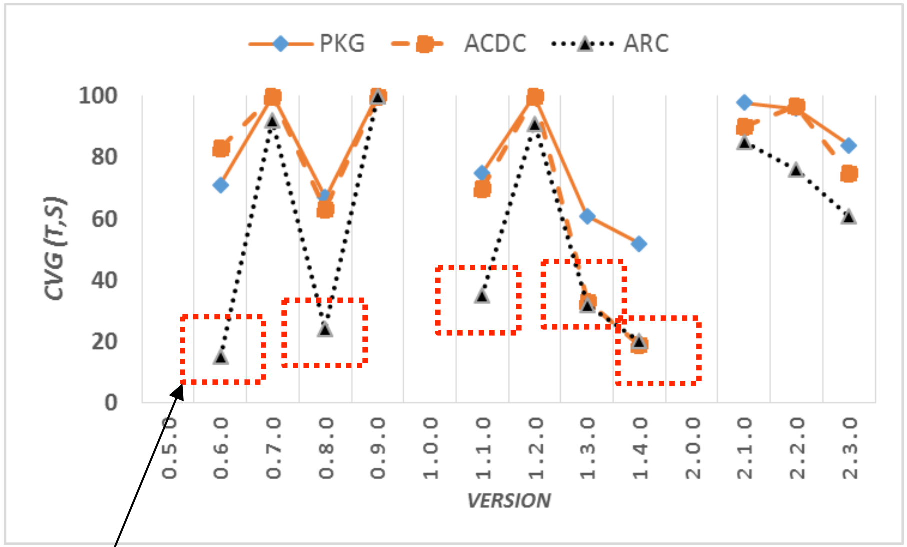
Major < MinMajor < Minor < Patch

System vs. Component Level

- Changes occur within components even when system's architectural structure remains relatively stable

Architectural similarity between minor versions of "Ivy"

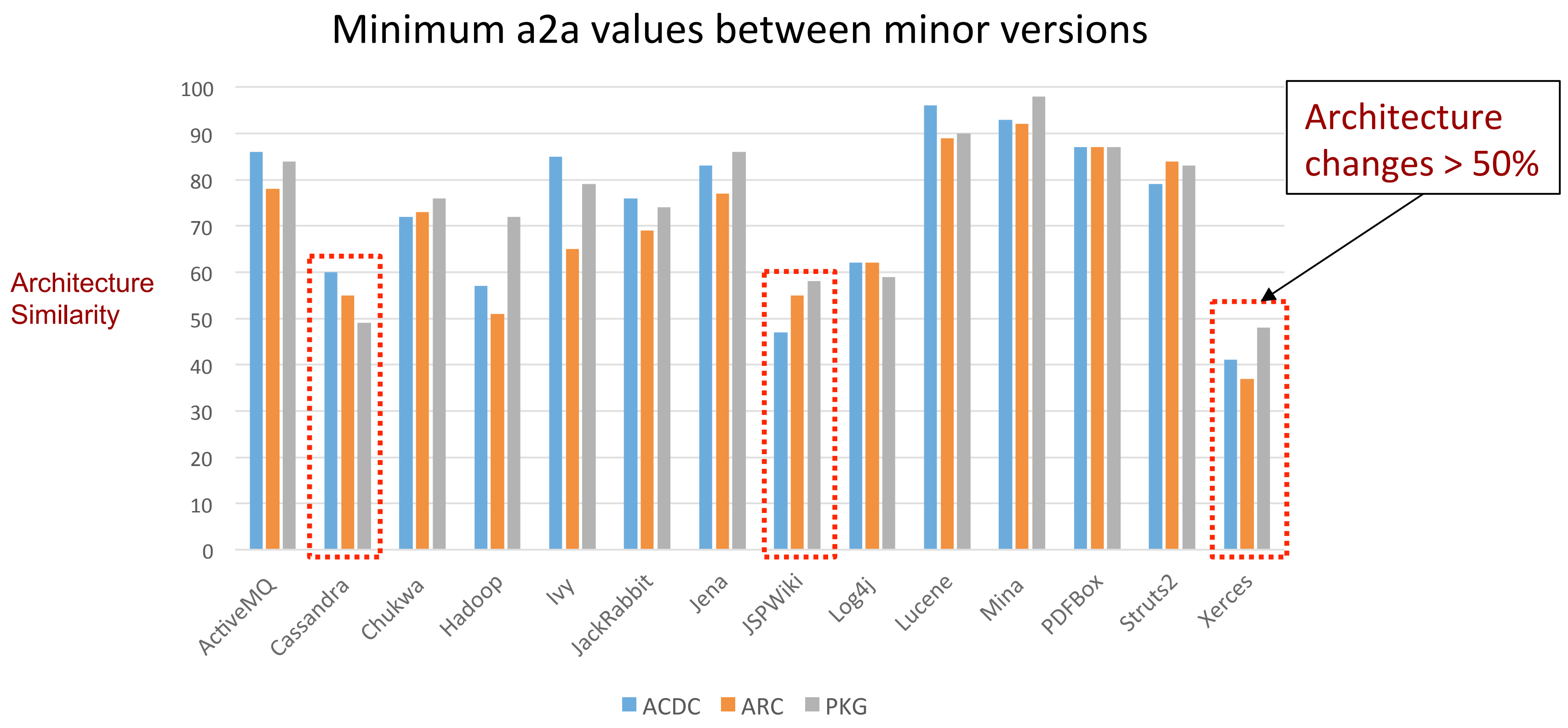




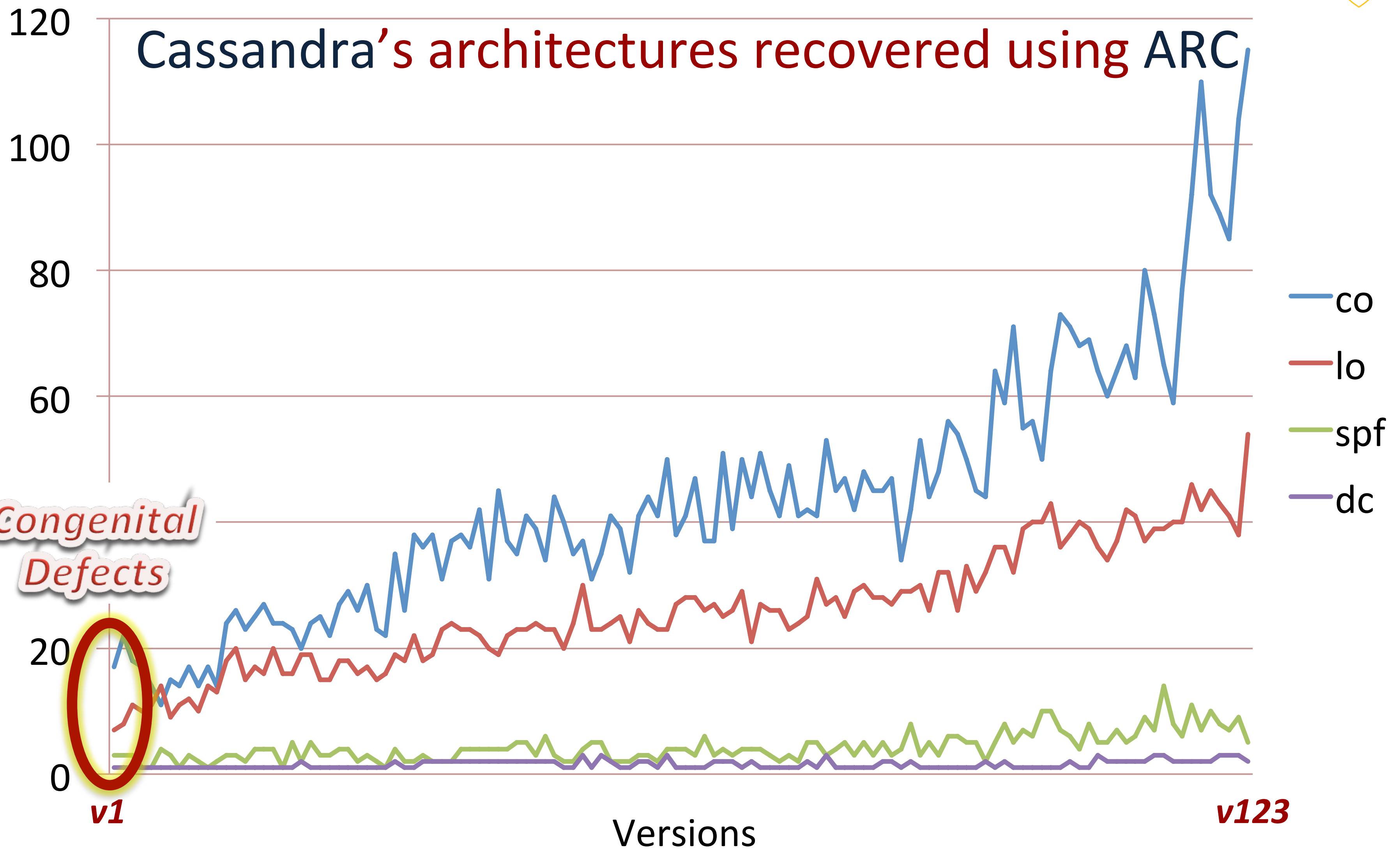
ARC view: architecture changes more than 80% within components

RQ3 – When Significant Change Occurs

- Dramatic architecture change can occur across minor versions

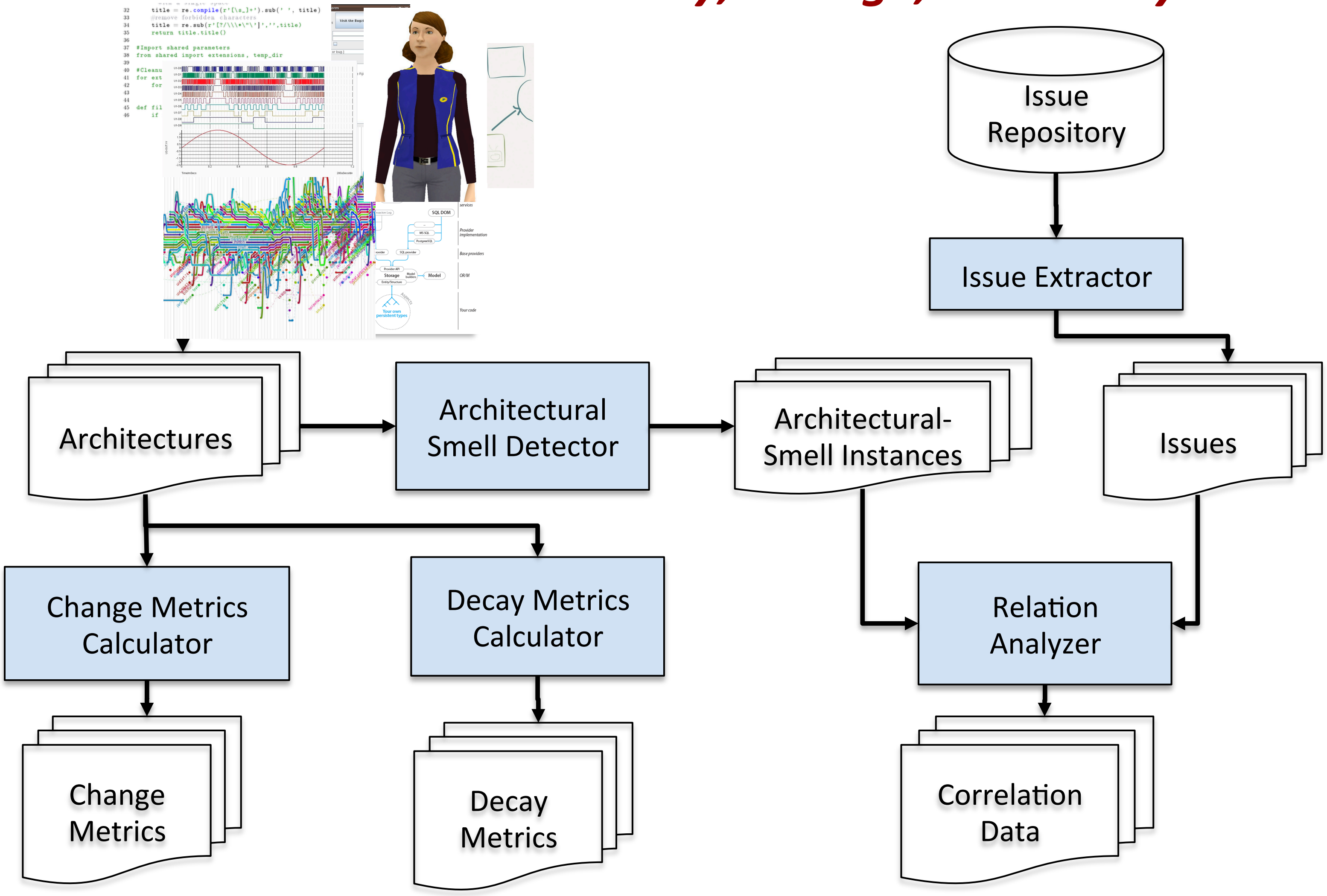


Architectural Decay



What We Don’t Know How To Do

ARCADE n.0 – Architecture Recovery, Change, and Decay Evaluator



```
    title = re.compile(r'[\s_]+').sub(' ', title)
    #remove forbidden characters
    title = re.sub(r'[?/\\\*\"\']','',title)
    return title.title()

#Import shared parameters
from shared import extensions, temp_dir

#Cleanu
for ext
    for


def fil
    if
```