

Refactoring Pipe-like Web Mashups for End-User Programmers

Kathryn T. Stolee and Sebastian Elbaum
University of Nebraska–Lincoln

May 25, 2011

Introduction

End User Programmers

People who engage in programming activities to support their hobbies and work.

Introduction

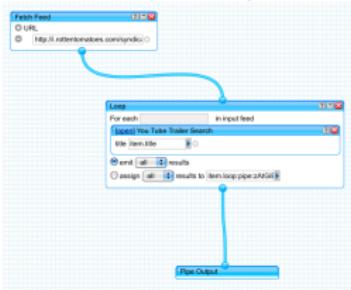
End User Programmers

People who engage in programming activities to support their hobbies and work.

	Professionals	End Users
Number in U.S. (2005)	3 million	55 million
Typical Education	C.S. Degree	Other Degree
Role of Programming	It's their job	It supports their job

Many Domains and Applications

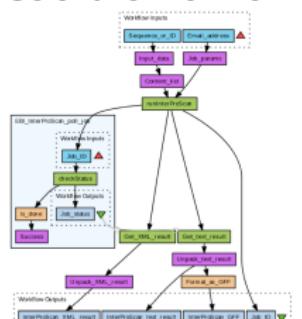
Web Mashups:



Educational Games:



Scientific Workflow:



Spreadsheets:

A screenshot of a web-based spreadsheet application. The interface includes a toolbar with file operations like "New", "Open", "Save", "Print", "Exit", and "Help". The main area shows a table titled "Search for houses in Willow Glen" with columns A through G. The table contains data for various companies, including their names and some numerical values. A sidebar on the left provides search and filter options, and a bottom panel displays a list of recent searches.

A screenshot of a desktop spreadsheet application showing a "Weekly Sales Sheet" for "Company A". The table has columns for "Week Ending", "Year", "Product", "Category", "Sales", "Profit", and "Margin". The data shows weekly sales figures for different products across categories. A "Grand Total" row at the bottom summarizes the sales for the week.

Week Ending	Year	Product	Category	Sales	Profit	Margin
10/13/07	2007	Product A	Category 1	\$10,000	\$1,000	10%
10/13/07	2007	Product B	Category 2	\$20,000	\$2,000	10%
10/13/07	2007	Product C	Category 3	\$30,000	\$3,000	10%
10/13/07	2007	Product D	Category 4	\$40,000	\$4,000	10%
10/13/07	2007	Product E	Category 5	\$50,000	\$5,000	10%
10/13/07	2007	Product F	Category 6	\$60,000	\$6,000	10%
10/13/07	2007	Product G	Category 7	\$70,000	\$7,000	10%
10/13/07	2007	Product H	Category 8	\$80,000	\$8,000	10%
10/13/07	2007	Product I	Category 9	\$90,000	\$9,000	10%
10/13/07	2007	Product J	Category 10	\$100,000	\$10,000	10%
Grand Total						
\$1,000,000						

Why Web Mashups?

Web Mashups

Applications that compose and manipulate existing data sources or services to create new data or service.

Why Study Mashups?

- Many environments (e.g., Apatar, DERI Pipes, IBM Mashup Center, Kivati, Yahoo! Pipes, ...)
- Potential impact
- Communities

Why Refactoring for Mashups?

- Mashup programs are littered with code smells
- Smells matter to end users

Stolee, K.T. and Elbaum, S. Refactoring Pipe-like Mashups for End-User Programmers. ICSE 2011

- Mashup languages are hard for end users to understand
- Maintenance in mashups is common: nearly 24% of Yahoo! Pipes mashups were modified after they were made public

Zang, N. and Rosson, M.B. Playing with information: How end users think about and integrate dynamic data. VL/HCC 2009

Stolee, K.T., Elbaum, S., and Sarma, A. End-User Programmers and their Communities: An Artifact-based Analysis. ESEM 2011

About Yahoo! Pipes

It's Popular

- Over 90,000 authors have created mashups
- Active community with message boards, tutorials, and literature
- Large public repository of sample code (nearly 100k pipes)

The screenshot shows a web browser window for the Yahoo! Pipes website. The title bar reads "Pipes: Rewrite the web". The main content area features a "Featured Pipe: NYC Apartment Near Something" with a thumbnail image of a city street at night. A sidebar on the right contains a quote from Paul Donnelly: "Ever wanted to find an apartment close to a park, a school or Whole Foods? Now you can... Created by Paul Donnelly (show me)". Below the featured pipe, there are sections for "About Pipes" and "Hot Pipes". The "About Pipes" section includes a brief description of what pipes are and how they work. The "Hot Pipes" section lists several popular pipes, such as "Flickr: Comments on my photos by other people" by bltrt, "Yahoo Finance Stock Quote Watch List Feed w/Chart" by Paul Donnelly, and "US population by state" by Jonathan. Each pipe entry includes a thumbnail, a brief description, and a link to more information.

About Yahoo! Pipes

It's Popular

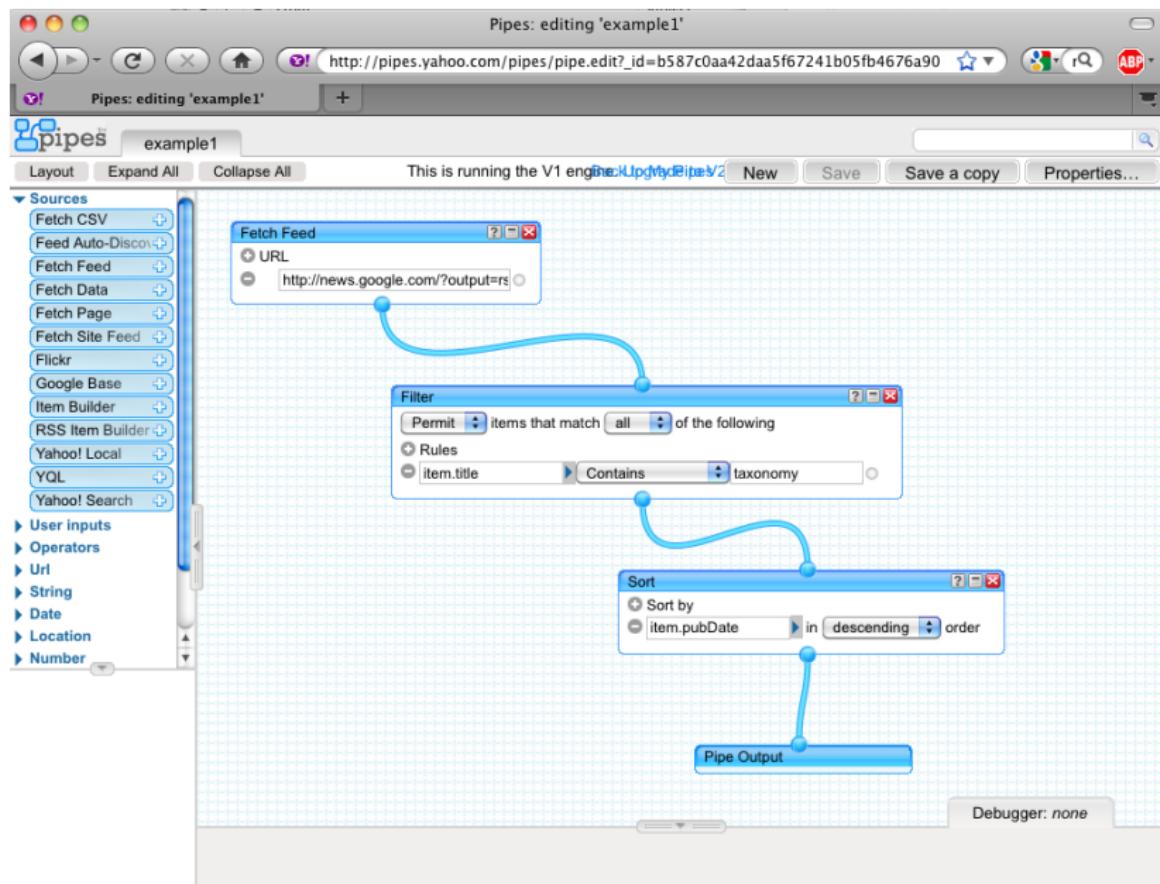
- Over 90,000 authors have created mashups
- Active community with message boards, tutorials, and literature
- Large public repository of sample code (nearly 100k pipes)

The screenshot shows a web browser window titled "Browse Pipes: Pipes containing the module 'fetch' (74065 results)". The URL is <http://pipes.yahoo.com/pipes/search?r=module:fetch>. The page header includes the Yahoo! Pipes logo, navigation links (Home, My Pipes, Browse, Discuss, Documentation, Create a pipe), and a search bar. A message at the top right says "You're logged in as kathyrommaset (logout)".

The main content area displays a list of 74065 pipes containing the 'fetch' module. Each entry includes a thumbnail icon, the pipe's name, the author's name (with a star icon indicating popularity), and a brief description. The pipes listed are:

- Title Mangler** by Ramblur: Lets you add text before and after the title in a feed. 397 clones.
- Add Feed Label to Each Item Title** by Randy (rockman): If you are aggregating multiple feeds, sometimes its nice to be able to tell which source an item is coming from. I like to add a tag or description of the blog or feed name to each item's title. This pipe is useful when inserted into another pipe and you can hard code the feed URL. 1307 clones.
- Title Mangler - Improved** by Christopher: I just took out some mistakes and exchanged Prefix/Postfix to make it more handy. Lets you add text before and after the title in a feed. Use for blanks between Prefix or Postfix and URL. Tags: feed feeds prefix editing improve +3... 114 clones.
- Update Maker for Lifestream** by Kingsley: If you would like to post lifestream updates like "Kingsley listened to 'Like You Used To' by J.J. Cale on Last.fm", then this is the pipe for you! Give it a feed url. Then give it a prefix (like "Kingsley listened to ") - it will be added in front of the title of each.... Sources: twitter.com 229 clones.
- YouTube tags to RSS** by Eric: A building block for the more advanced YouTube filter Pipe. Tags: rss youtube subscribe Sources: youtube.com 1034 clones.
- Search E7TV for TV Shows** by ChesterC: ...

About Yahoo! Pipes



About Yahoo! Pipes

Pipes: editing 'example1'

Pipes: editing 'example1' [New](#) [Save](#) [Save a copy](#) [Properties...](#)

This is running the V1 engine. [Upgrading to V2](#)

Sources

- Fetch CSV
- Feed Auto-Discov.
- Fetch Feed
- Fetch Data
- Fetch Page
- Fetch Site Feed
- Flickr
- Google Base
- Item Builder
- RSS Item Builder
- Yahoo! Local
- YQL
- Yahoo! Search

User Inputs

Operators

Url

String

Date

Location

Number

Fetch Feed

URL
http://news.google.com/?output=rss

Filter

Permit items that match all of the following

Rules

item.title Contains taxonomy

Sort

Sort by item.pubDate in descending order

Pipe Output

Debugger: none

```
graph TD; Fetch[Fetch Feed] --> Filter[Filter]; Filter --> Sort[Sort]; Sort --> Output[Pipe Output]
```

About Yahoo! Pipes

Pipes: editing 'example1'

Pipes: editing 'example1' [New](#) [Save](#) [Save a copy](#) [Properties...](#)

This is running the V1 engine. [Switch to V2](#)

Sources

- Fetch CSV
- Feed Auto-Discov.
- Fetch Feed
- Fetch Data
- Fetch Page
- Fetch Site Feed
- Flickr
- Google Base
- Item Builder
- RSS Item Builder
- Yahoo! Local
- YQL
- Yahoo! Search

User Inputs

Operators

Url

String

Date

Location

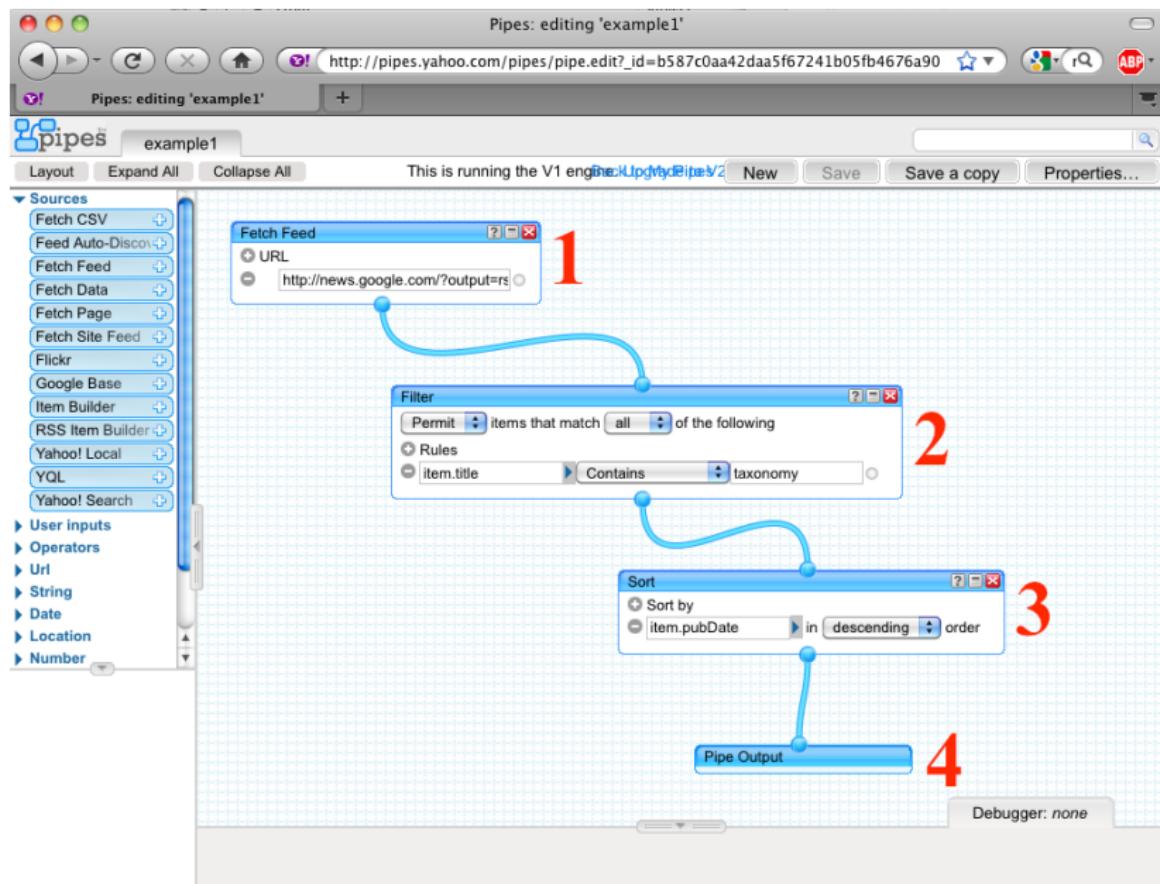
Number

The screenshot shows a Yahoo! Pipes workflow titled 'example1'. The process starts with a 'Fetch Feed' operator, which retrieves data from 'http://news.google.com/?output=rss'. This feed is then processed by a 'Filter' operator, which permits items that match the rule 'item.title Contains taxonomy'. Finally, the data is sorted by 'item.pubDate' in descending order, and the results are outputted. The entire pipe is currently running on the V1 engine.

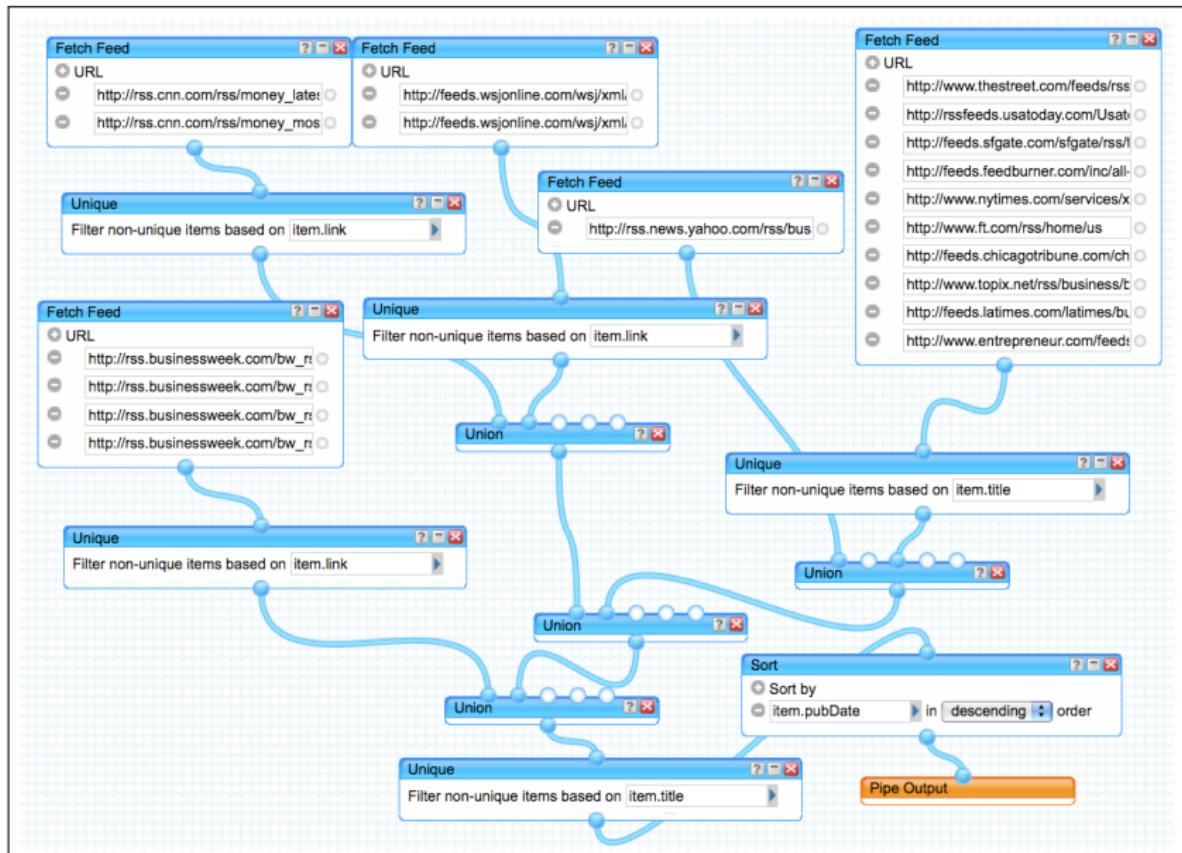
```
graph TD; Fetch[Fetch Feed] --> Filter[Filter]; Filter --> Sort[Sort]; Sort --> Output[Pipe Output]
```

Debugger: none

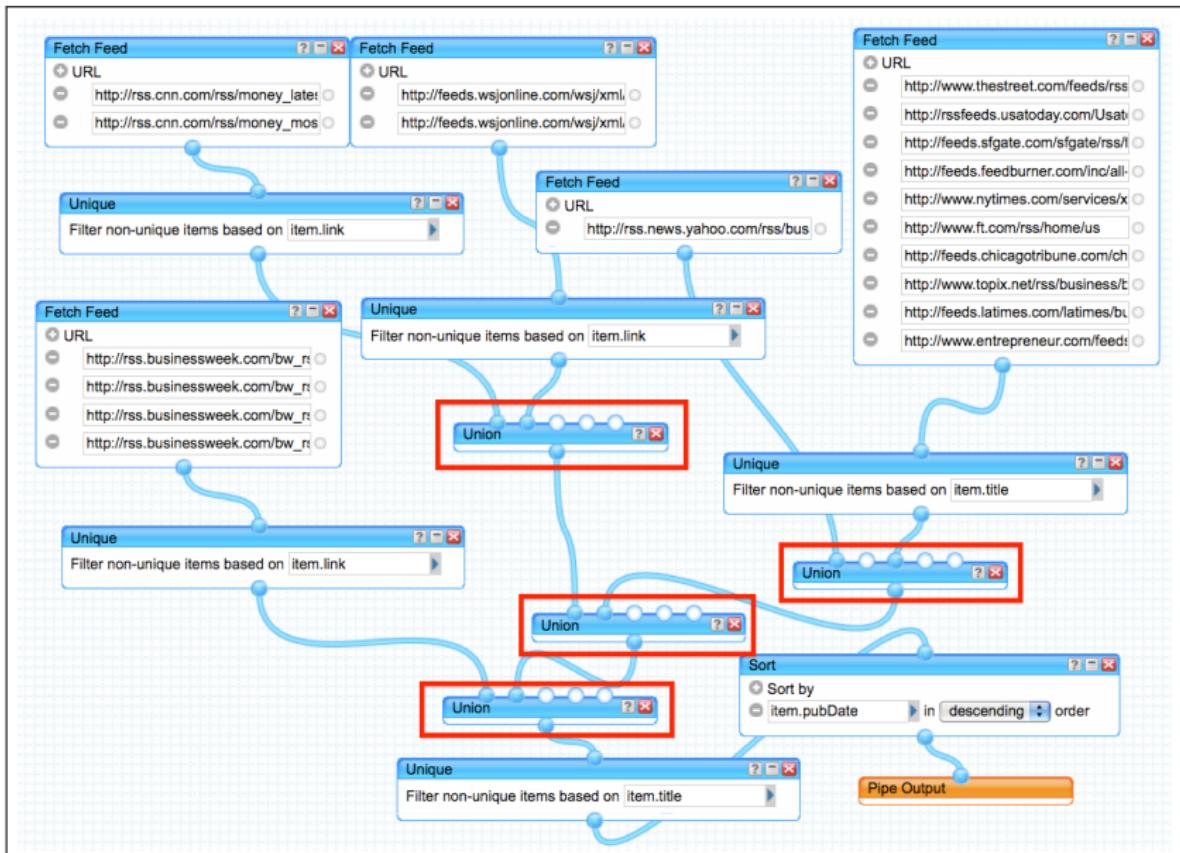
About Yahoo! Pipes



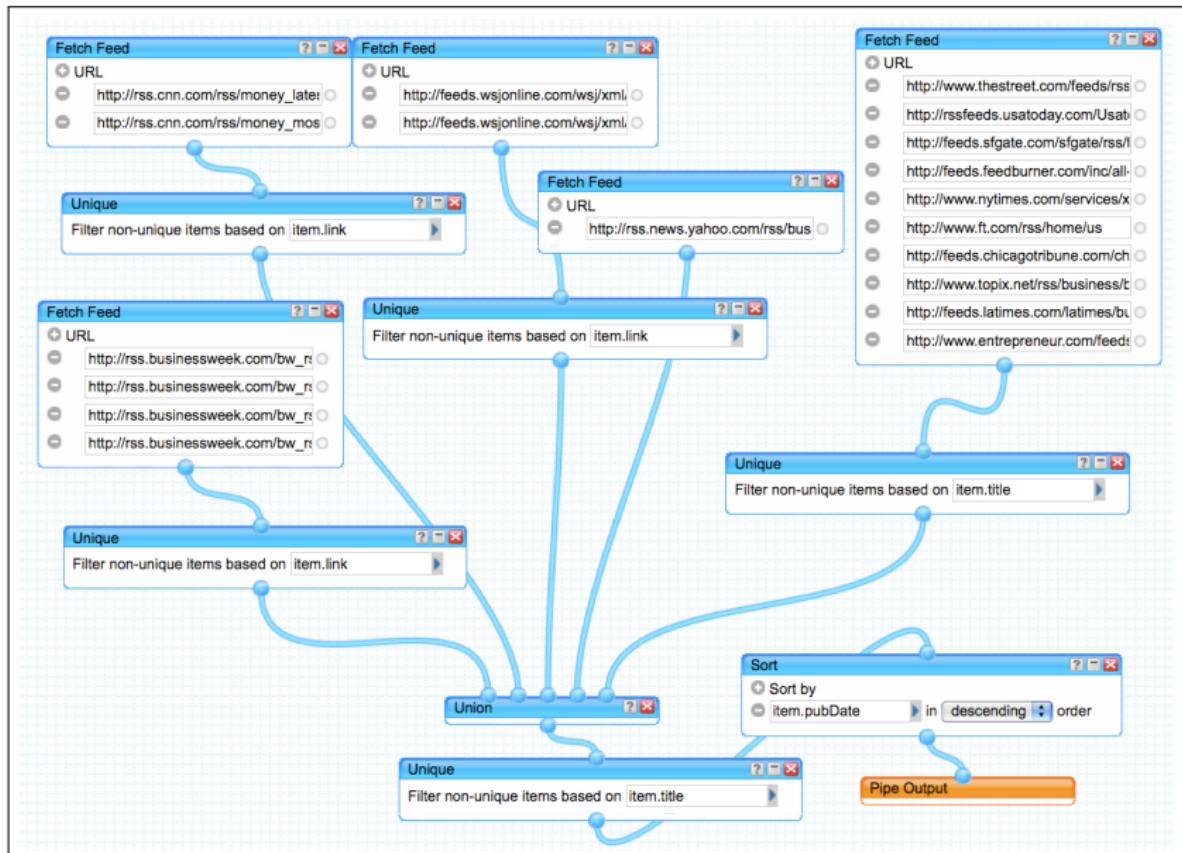
Motivating Example



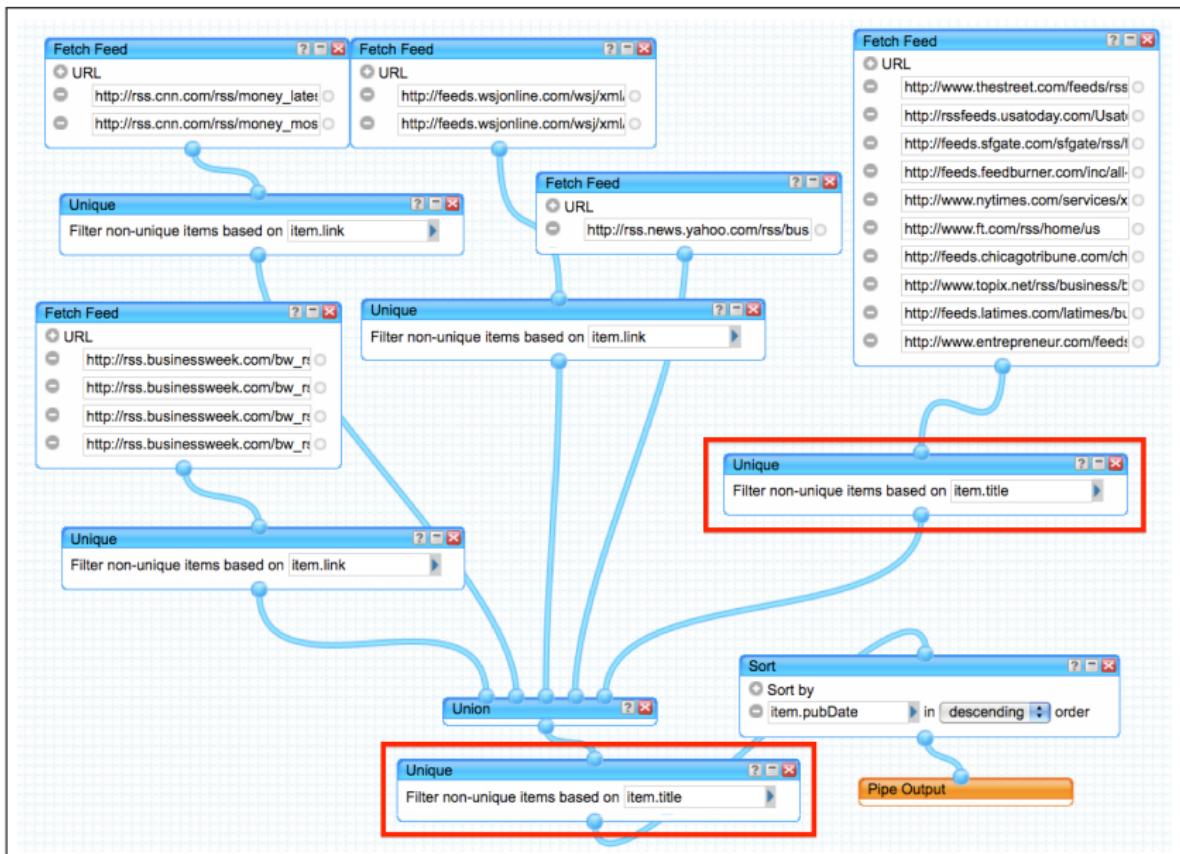
Motivating Example



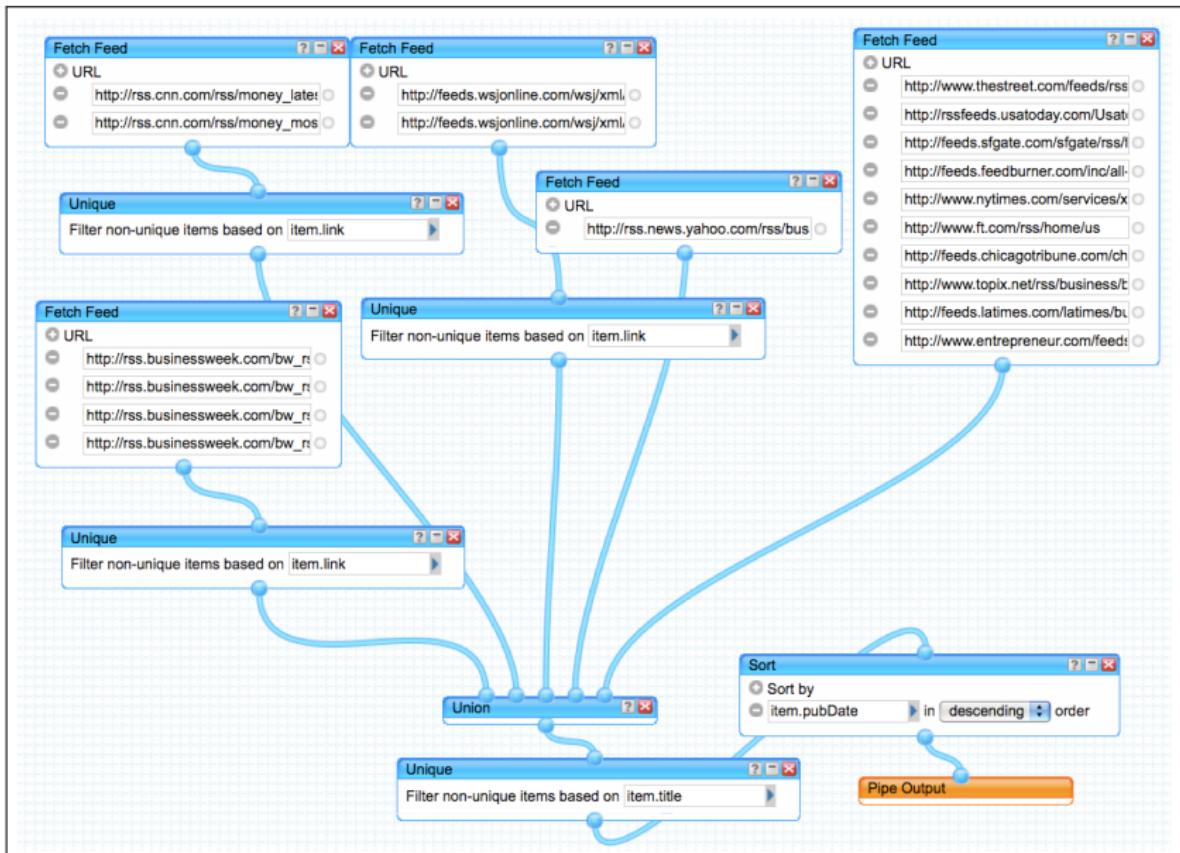
Motivating Example



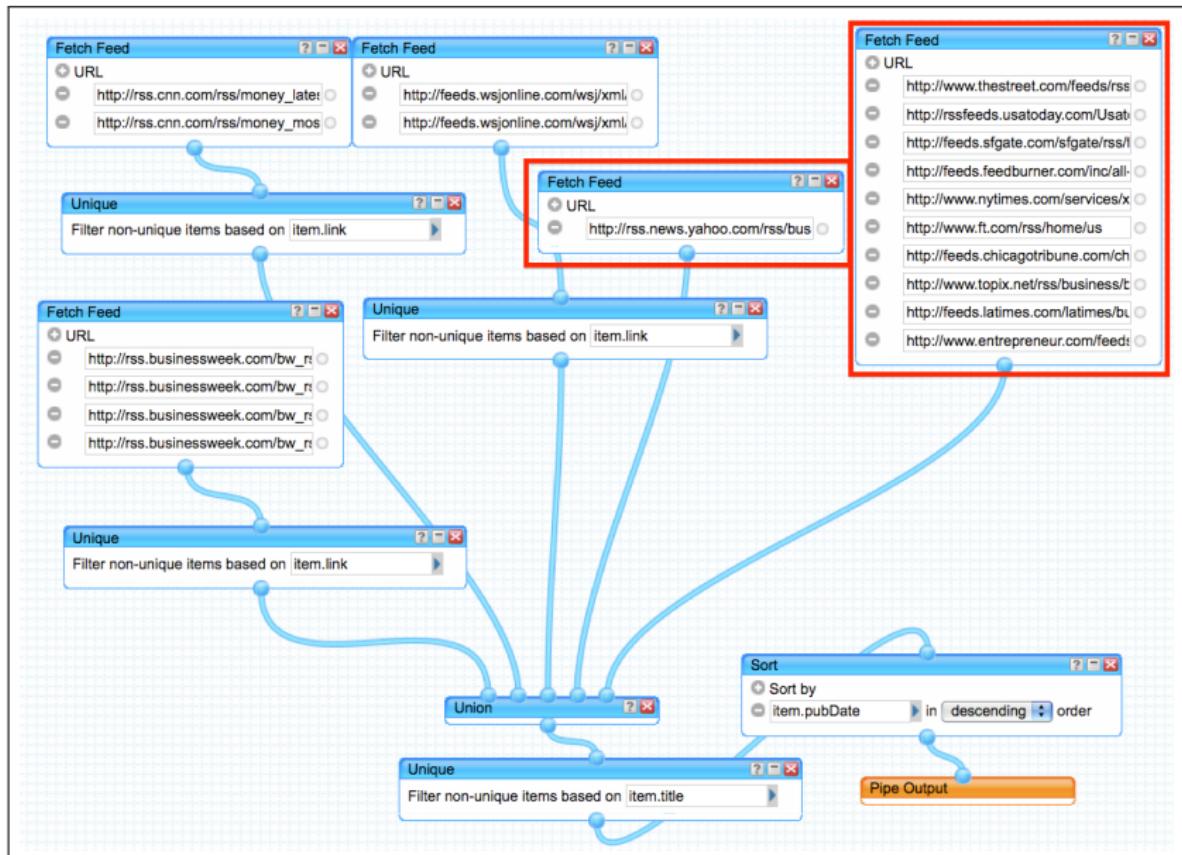
Motivating Example



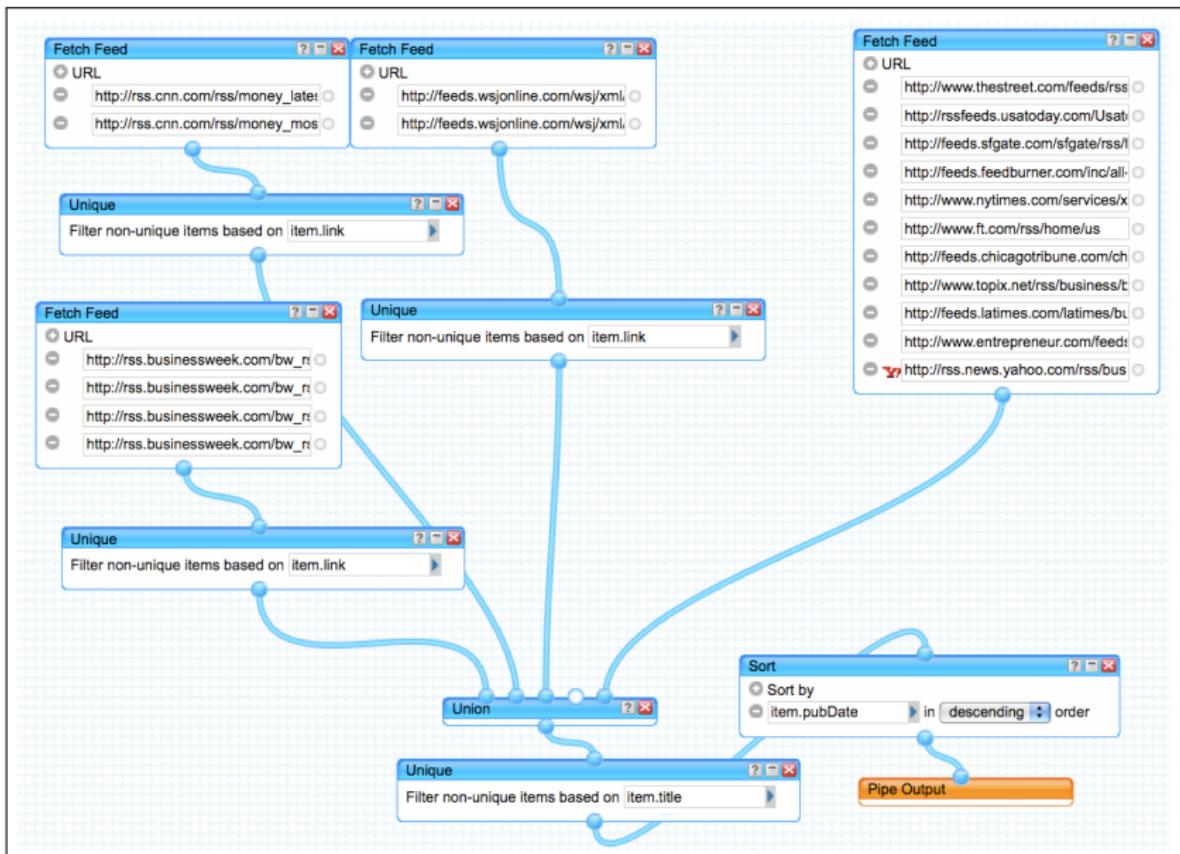
Motivating Example



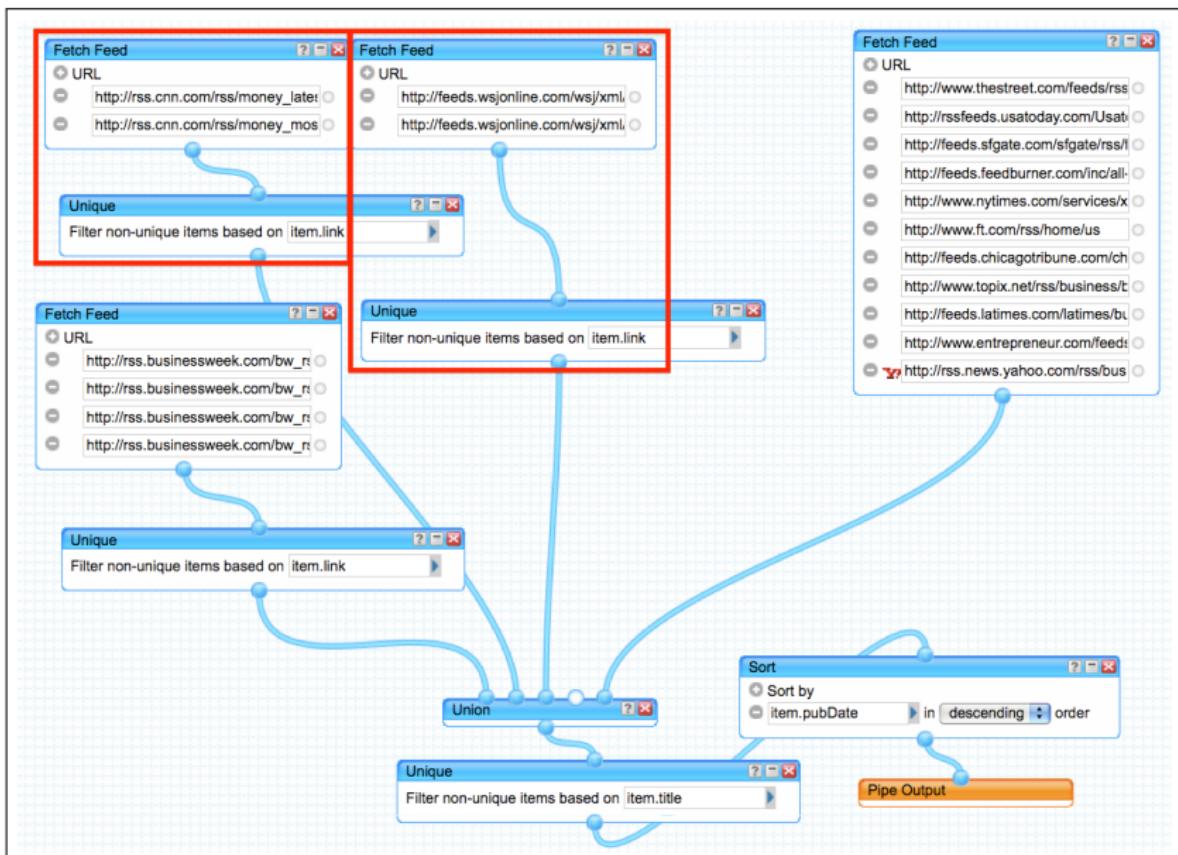
Motivating Example



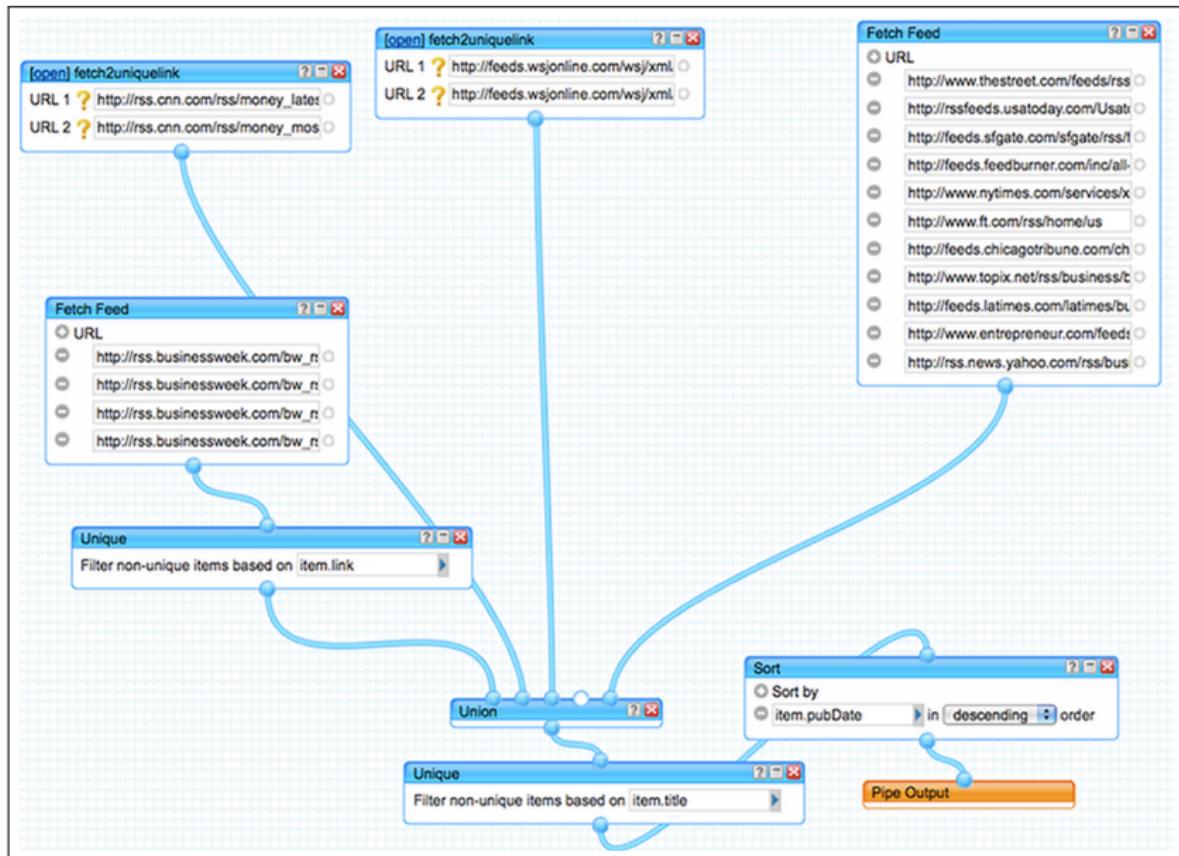
Motivating Example



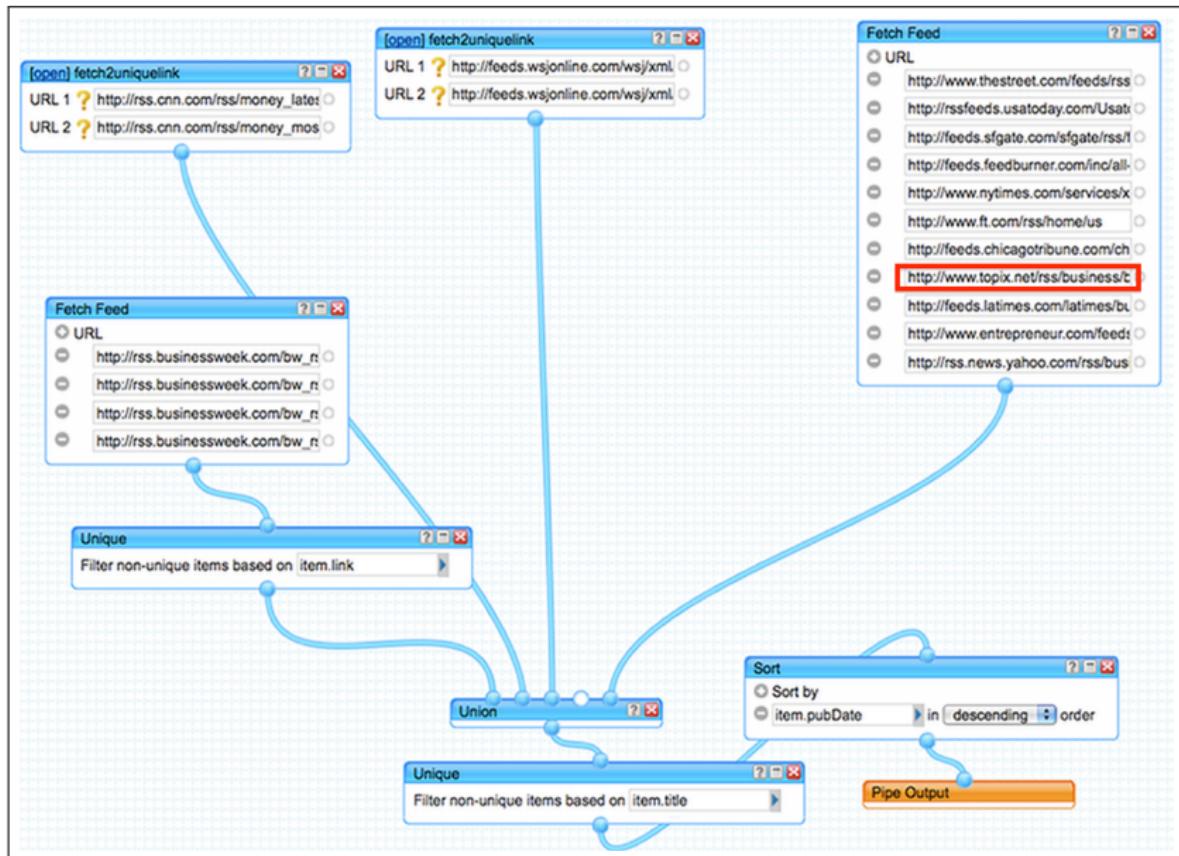
Motivating Example



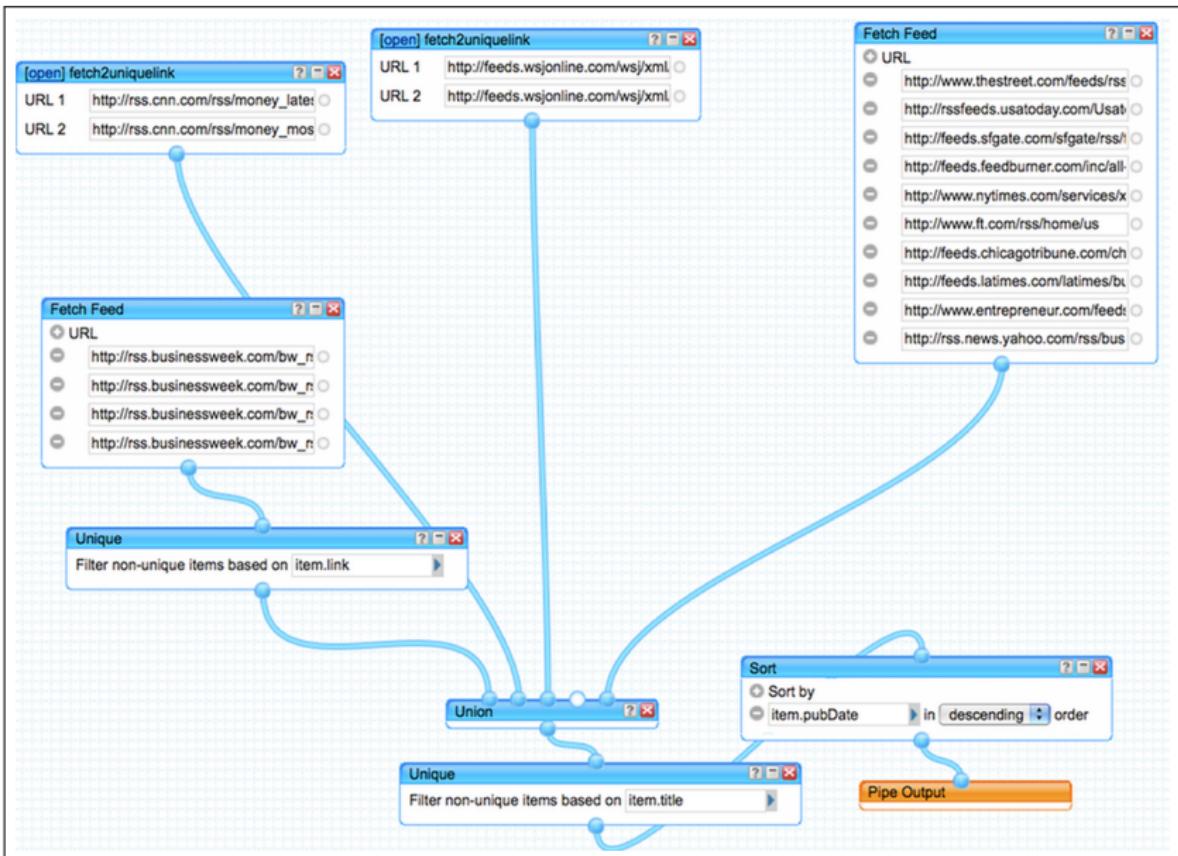
Motivating Example



Motivating Example

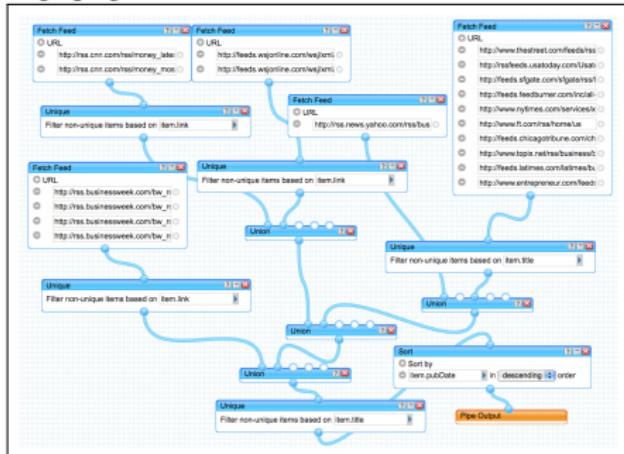


Motivating Example

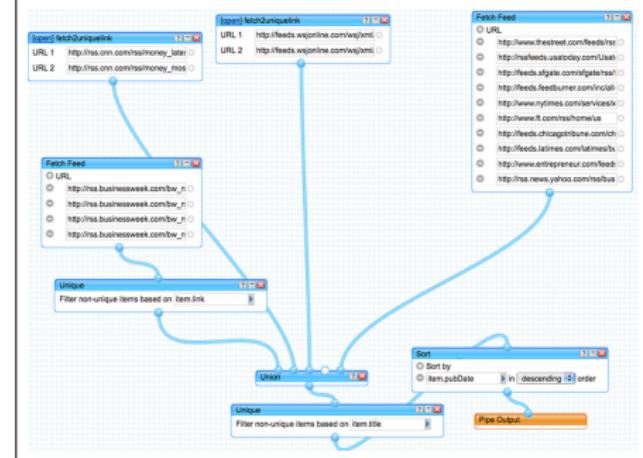


Motivating Example: Before and After

Before



After

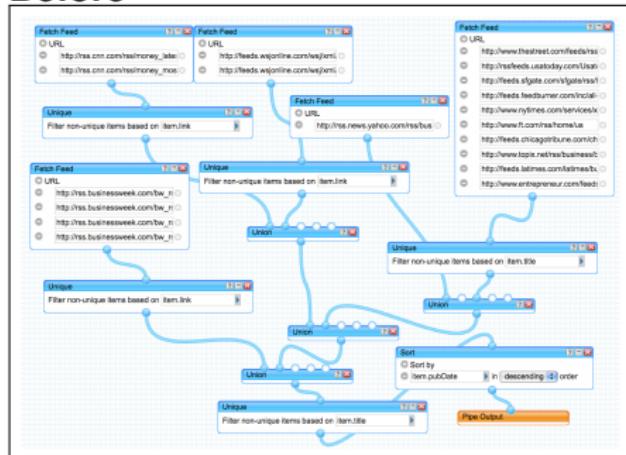


- Redundant structures
- Broken data sources

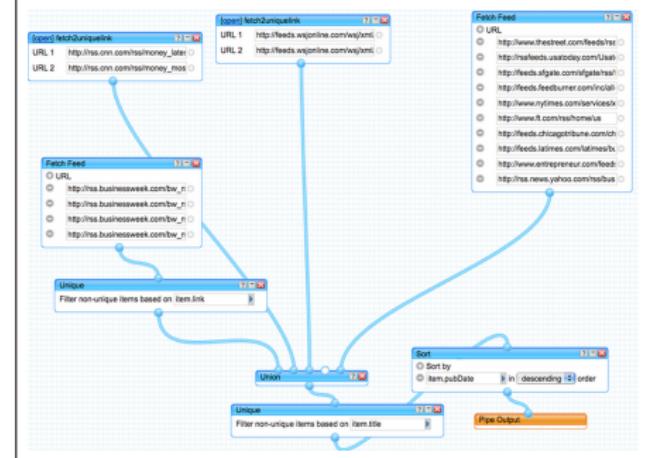
- Less complex
- All valid sources

Motivating Example: Before and After

Before



After



- Redundant structures
- Broken data sources

- Less complex
- All valid sources

Issue: Deficiencies are propagated through reuse

- Out of 8,000+ pipes, 66% had been reused
- 810 pipes (10%) had been reused *and* reference invalid sources

Definitions

Definitions

Code Smell

Program characteristics that identify deficiencies in code

Definitions

Code Smell

Program characteristics that identify deficiencies in code

Refactoring

A semantic preserving transformation on a program

Definitions

Code Smell

Program characteristics that identify deficiencies in code

Refactoring

A semantic preserving transformation on a program

We seek to better support **end user programmers** by applying the software engineering concepts of **code smells** and **refactorings** to **web mashups**.

Smells in Mashups

Smell Type	Smells
Laziness	Noisy Module Unnecessary Module Unnecessary Abstraction
Redundancy	Duplicate Strings Duplicate Modules Isomorphic Paths
Environmental	Deprecated Module Invalid Source
Population-Based	Non-Conforming Ordering Global Isomorphic Paths

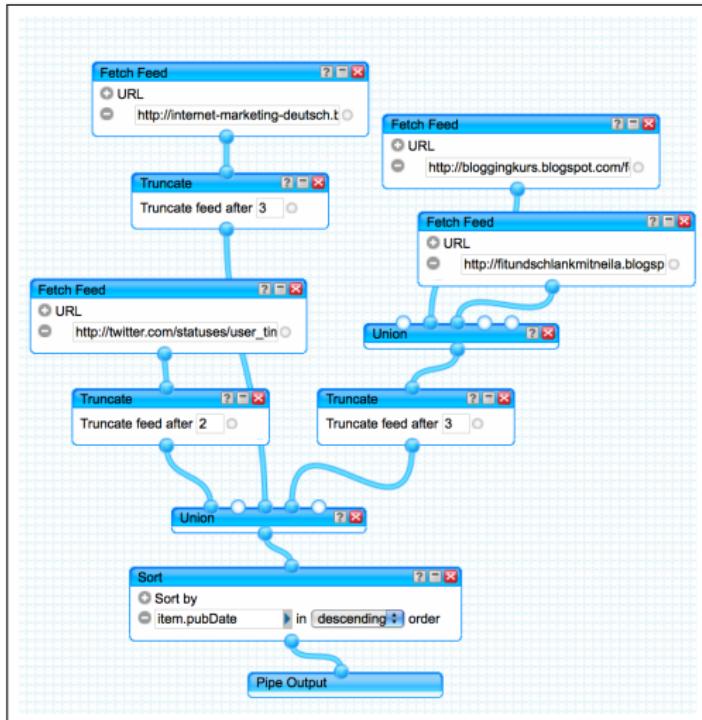
Redundancy: Duplicate Modules

Joined Generators

Similar modules appearing in certain patterns may be redundant and candidate for consolidation.

Why is this Smelly?

Module redundancies add unnecessary complexity



Redundancy: Duplicate Modules

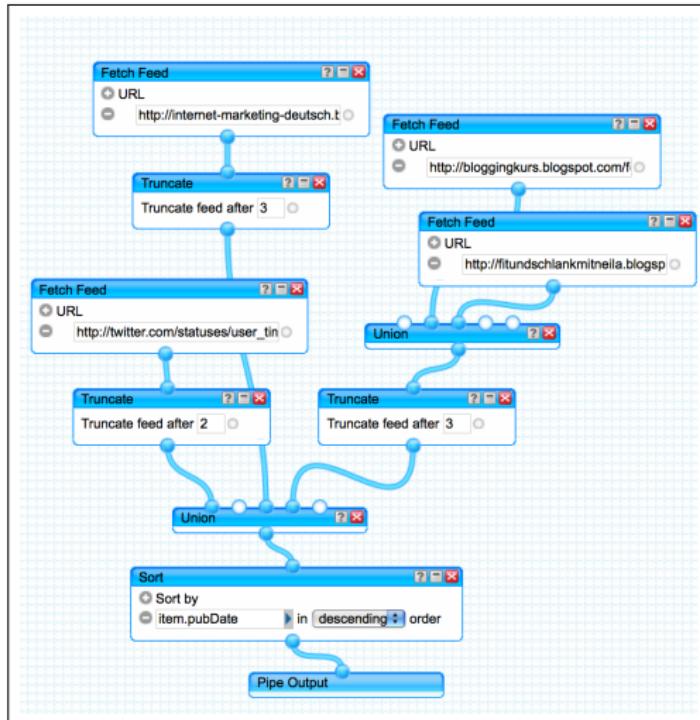
Joined Generators

Similar modules appearing in certain patterns may be redundant and candidate for consolidation.

Why is this Smelly?

Module redundancies add unnecessary complexity

Definition

$$\exists m, n \in \mathcal{M} \mid m \neq n \wedge \\ m.name = n.name \wedge \\ gen(m) \wedge gen(n) \wedge \\ connected_to_union(m, n)$$


Redundancy: Duplicate Modules

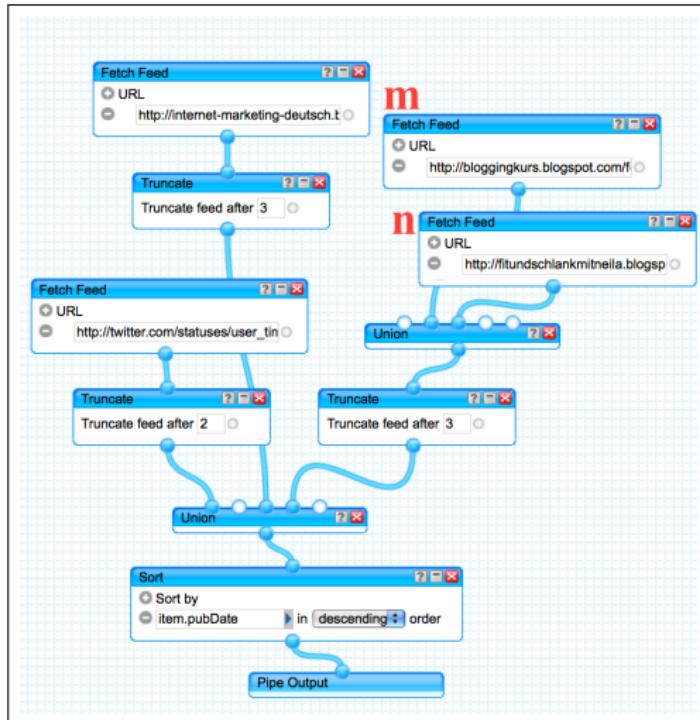
Joined Generators

Similar modules appearing in certain patterns may be redundant and candidate for consolidation.

Why is this Smelly?

Module redundancies add unnecessary complexity

Definition

$$\exists m, n \in \mathcal{M} \mid m \neq n \wedge$$


Redundancy: Duplicate Modules

Joined Generators

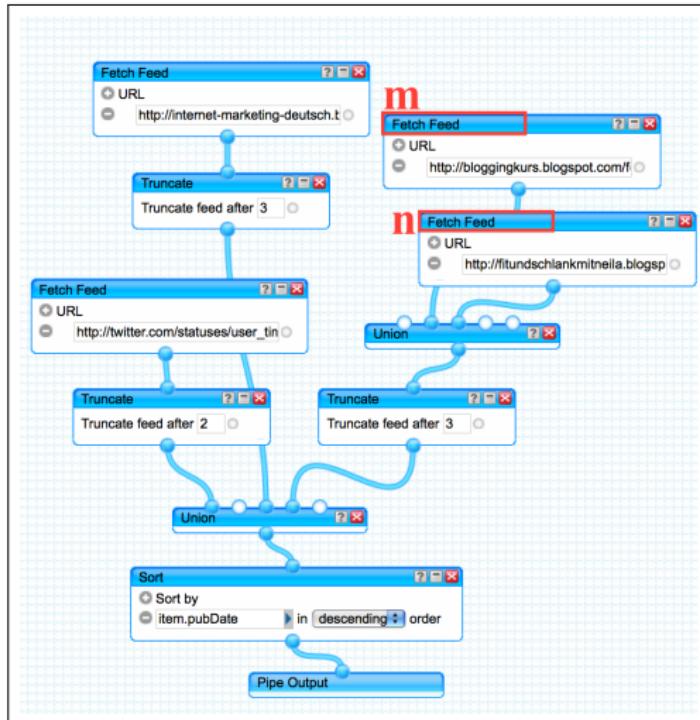
Similar modules appearing in certain patterns may be redundant and candidate for consolidation.

Why is this Smelly?

Module redundancies add unnecessary complexity

Definition

$$\exists m, n \in \mathcal{M} \mid m \neq n \wedge \\ m.name = n.name \wedge$$



Redundancy: Duplicate Modules

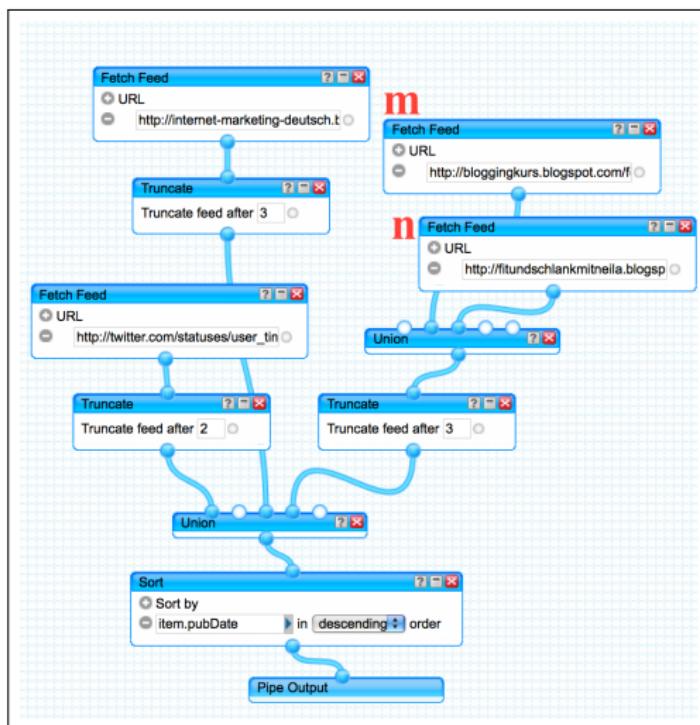
Joined Generators

Similar modules appearing in certain patterns may be redundant and candidate for consolidation.

Why is this Smelly?

Module redundancies add unnecessary complexity

Definition

$$\exists m, n \in \mathcal{M} \mid m \neq n \wedge \\ m.name = n.name \wedge \\ gen(m) \wedge gen(n) \wedge$$


Redundancy: Duplicate Modules

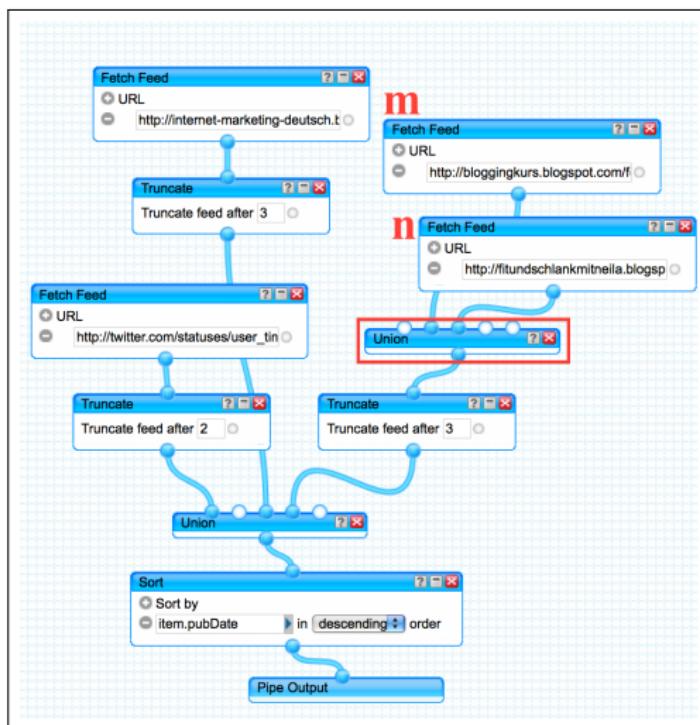
Joined Generators

Similar modules appearing in certain patterns may be redundant and candidate for consolidation.

Why is this Smelly?

Module redundancies add unnecessary complexity

Definition

$$\exists m, n \in \mathcal{M} \mid m \neq n \wedge \\ m.name = n.name \wedge \\ gen(m) \wedge gen(n) \wedge \\ connected_to_union(m, n)$$


Redundancy: Duplicate Modules

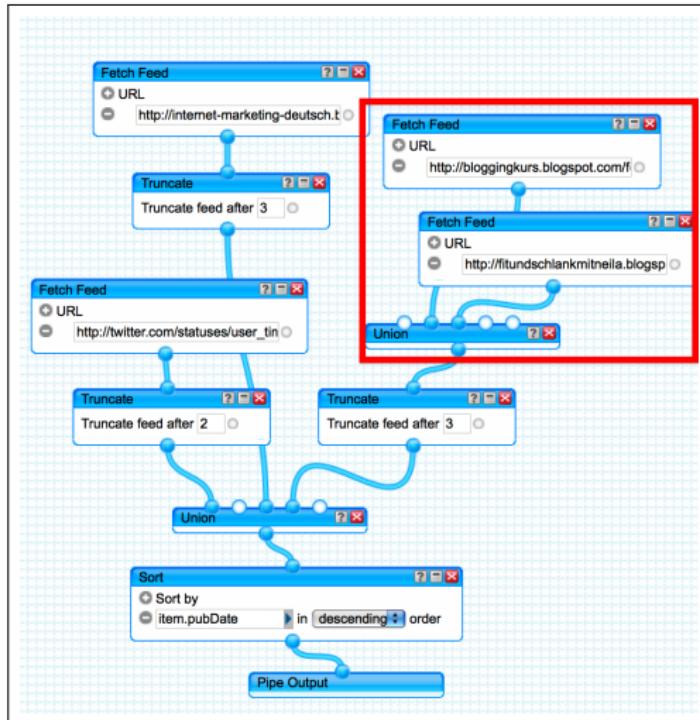
Joined Generators

Similar modules appearing in certain patterns may be redundant and candidate for consolidation.

Why is this Smelly?

Module redundancies add unnecessary complexity

Definition

$$\exists m, n \in \mathcal{M} \mid m \neq n \wedge \\ m.name = n.name \wedge \\ gen(m) \wedge gen(n) \wedge \\ connected_to_union(m, n)$$


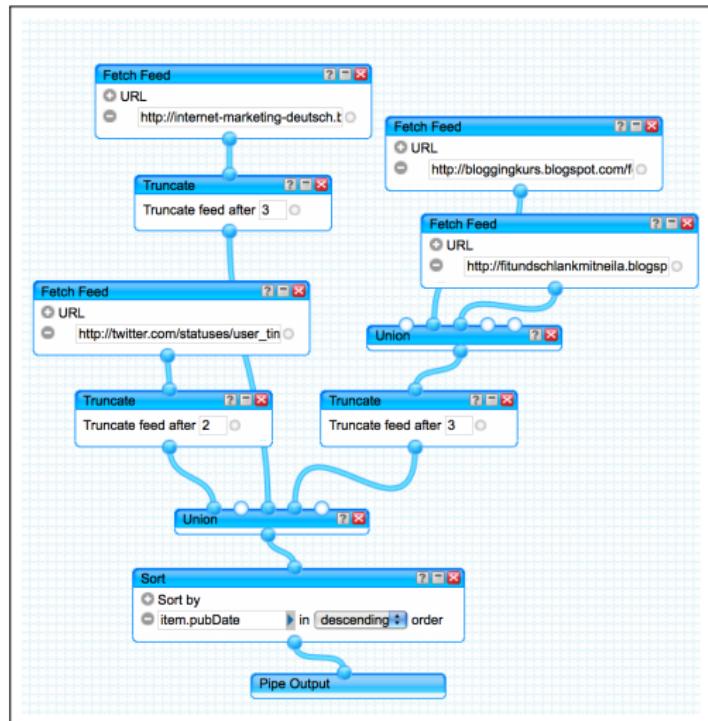
Redundancy: Duplicate Strings

Duplicate Strings:

A constant string that is used in at least n wireable fields in at least two modules.

Why is this Smelly?

Missed opportunity for abstraction



Redundancy: Duplicate Strings

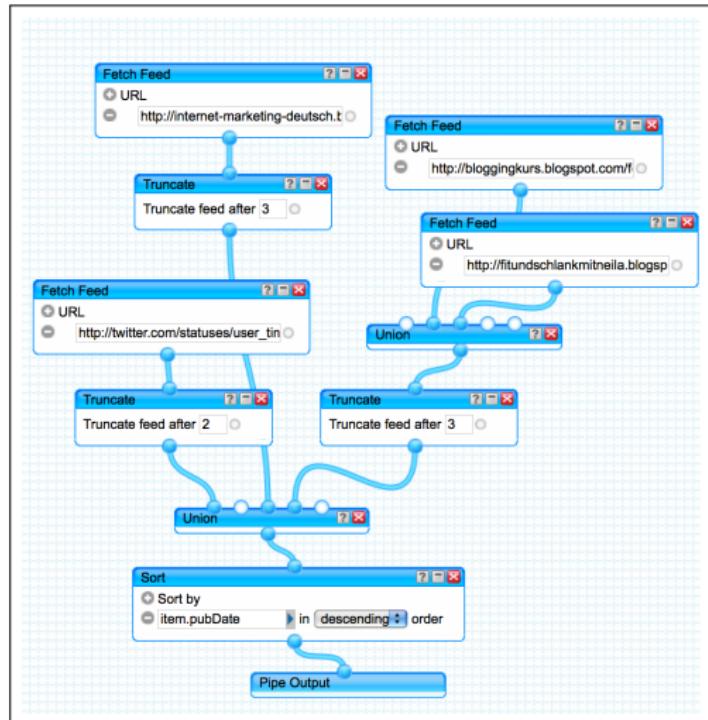
Duplicate Strings:

A constant string that is used in at least n wireable fields in at least two modules.

Why is this Smelly?

Missed opportunity for abstraction

Definition

$$\exists f, g \in \mathcal{F} \mid f \neq g \wedge \\ \text{same_field_values}(f, g) \wedge \\ \text{owner}(f) \neq \text{owner}(g)$$


Redundancy: Duplicate Strings

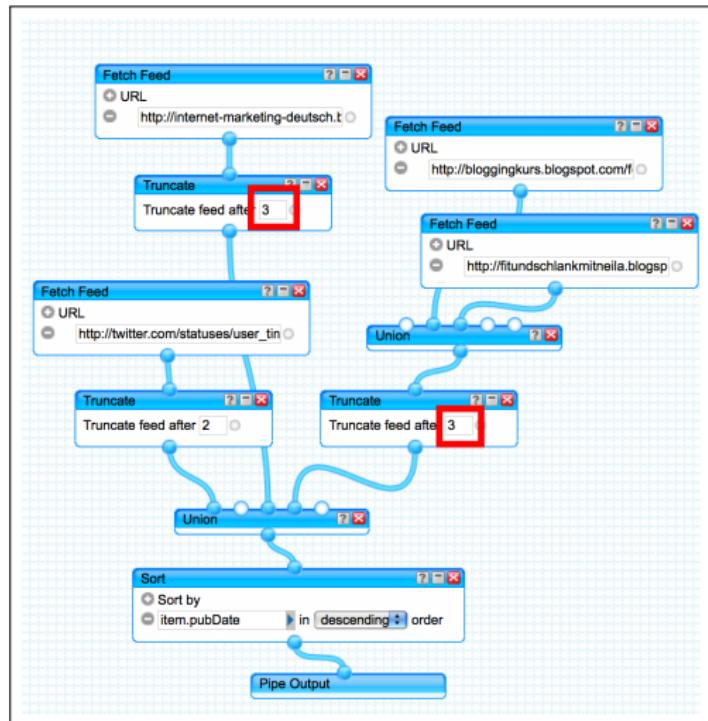
Duplicate Strings:

A constant string that is used in at least n wireable fields in at least two modules.

Why is this Smelly?

Missed opportunity for abstraction

Definition

$$\exists f, g \in \mathcal{F} \mid f \neq g \wedge \\ \text{same_field_values}(f, g) \wedge \\ \text{owner}(f) \neq \text{owner}(g)$$


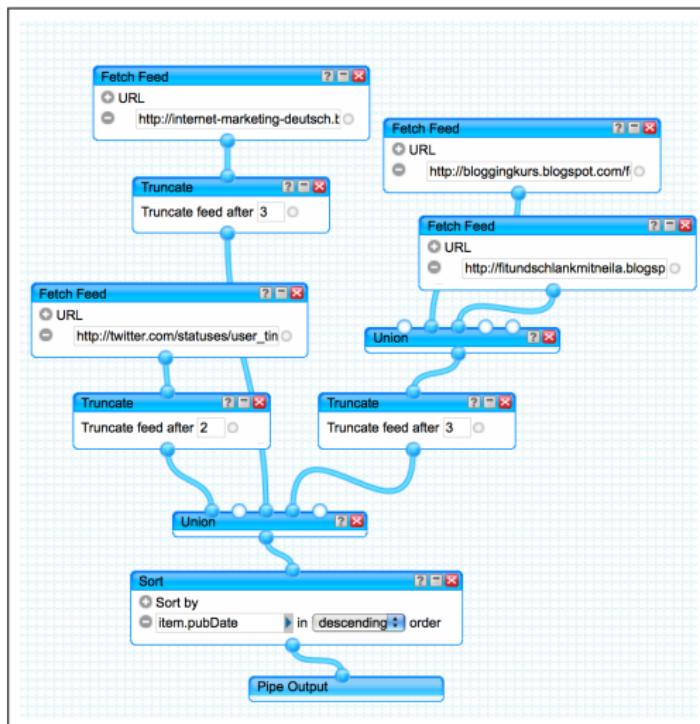
Population-Based: Global Isomorphic Paths

Global Isomorphic Path

A path in a pipe that is isomorphic to other paths that appear elsewhere in the population

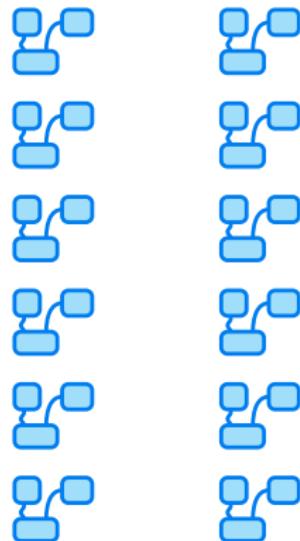
Why is this Smelly?

It's a missed opportunity for standardization and abstraction



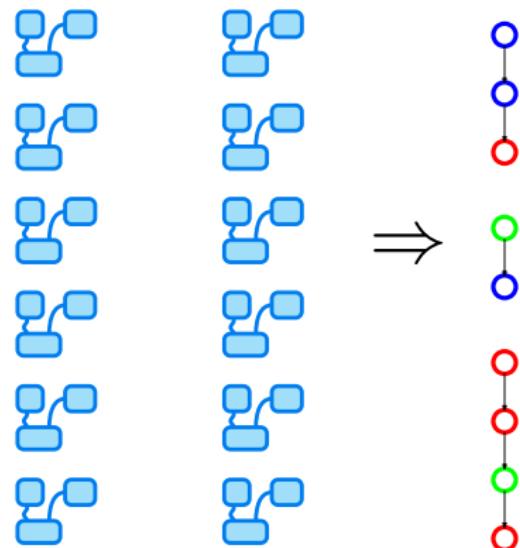
Population-Based Refactorings

Pipes Population



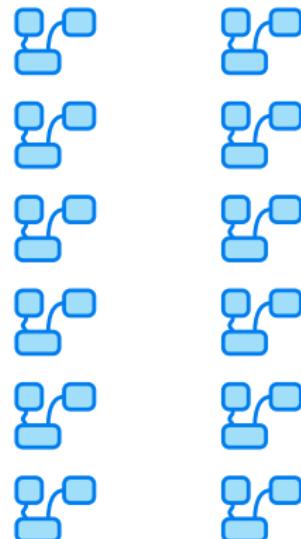
Population-Based Refactorings

Pipes Population Common Paths



Population-Based Refactorings

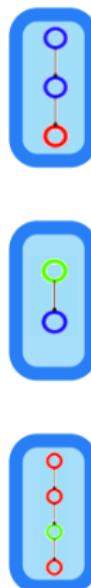
Pipes Population



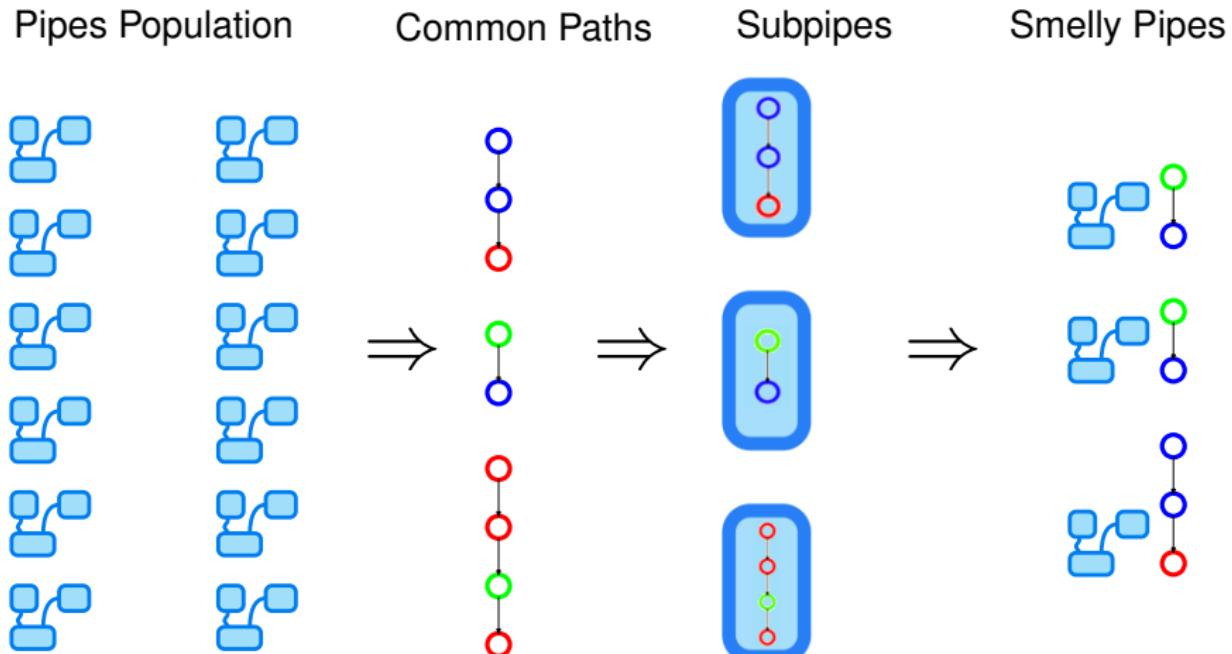
Common Paths



Subpipes



Population-Based Refactorings



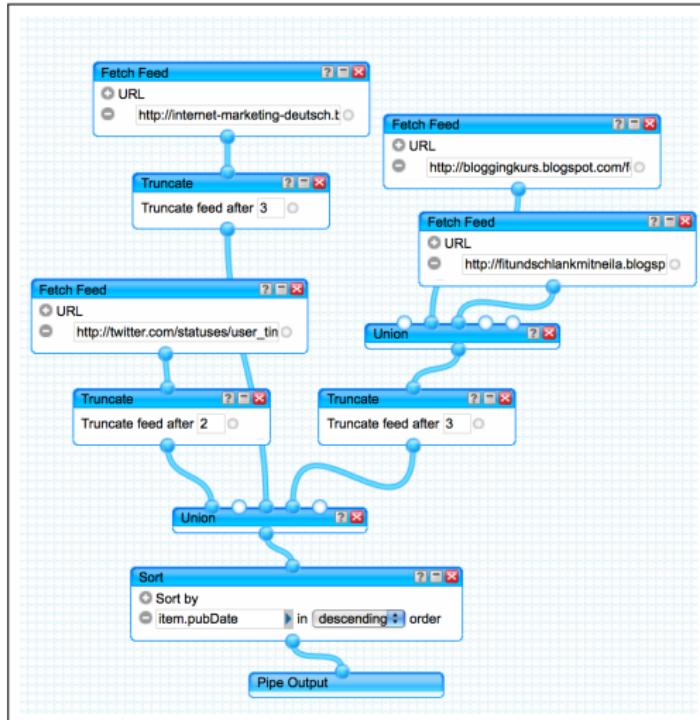
Population-Based: Global Isomorphic Paths

Global Isomorphic Path

A path in a pipe that is isomorphic to other paths that appear elsewhere in the population

Why is this Smelly?

It's a missed opportunity for standardization and abstraction



Population-Based: Global Isomorphic Paths

Global Isomorphic Path

A path in a pipe that is isomorphic to other paths that appear elsewhere in the population

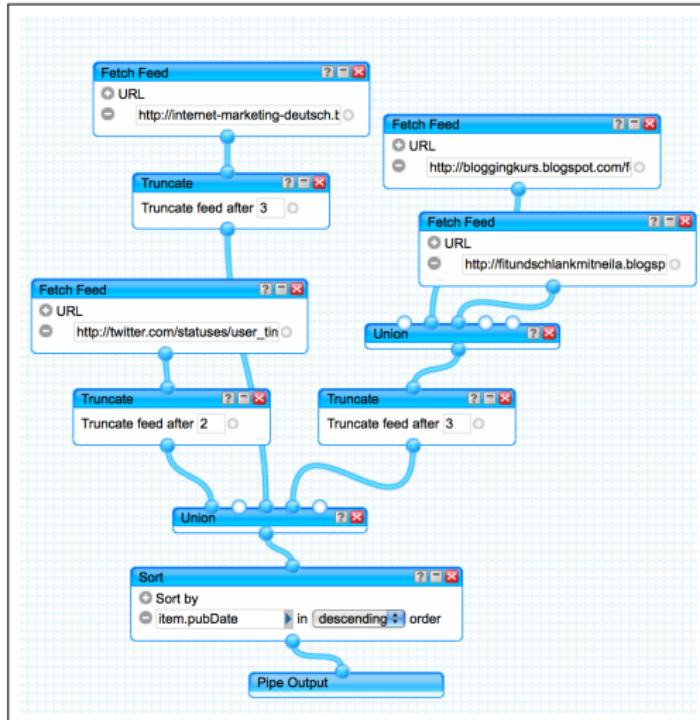
Why is this Smelly?

It's a missed opportunity for standardization and abstraction

Definition

Given a pool of global paths $PGPaths$, a pipe P has this smell if:

$$\exists p \in P, \exists p' \in PGPaths \mid p' \text{ is isomorphic to } p$$



Population-Based: Global Isomorphic Paths

Global Isomorphic Path

A path in a pipe that is isomorphic to other paths that appear elsewhere in the population

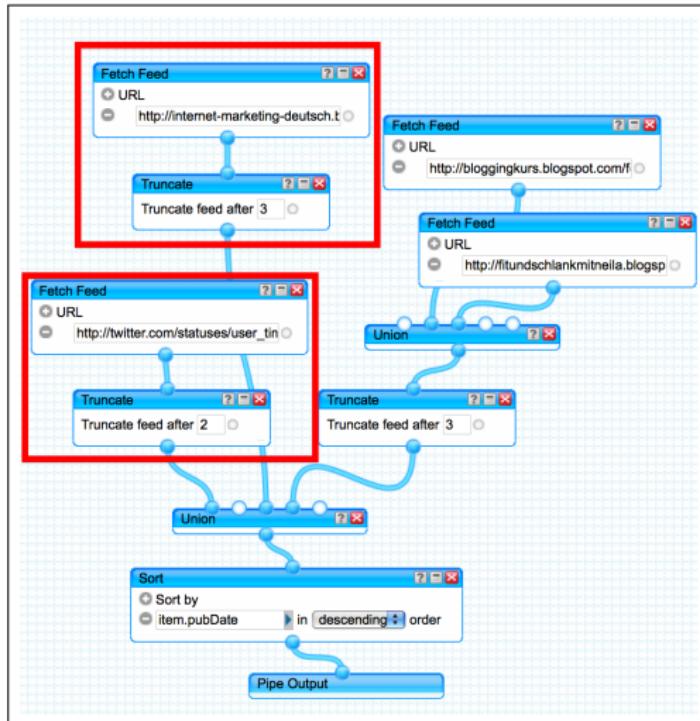
Why is this Smelly?

It's a missed opportunity for standardization and abstraction

Definition

Given a pool of global paths $PGPaths$, a pipe P has this smell if:

$$\exists p \in P, \exists p' \in PGPaths \mid p' \text{ is isomorphic to } p$$



End User Study

Research Question I

Are pipes with smells **less preferable** than pipes without smells?

Research Question II

Are pipes with smells **less understandable** than pipes without smells?

End User Study

Empirical Study with 14 participants

Research Question I

Are pipes with smells **less preferable** than pipes without smells?

Research Question II

Are pipes with smells **less understandable** than pipes without smells?

End User Study

Empirical Study with 14 participants

Research Question I

Are pipes with smells **less preferable** than pipes without smells?

Yes: 63% **No:** 24% **Same:** 13%. (8 tasks)

Research Question II

Are pipes with smells **less understandable** than pipes without smells?

End User Study

Empirical Study with 14 participants

Research Question I

Are pipes with smells **less preferable** than pipes without smells?

Yes: 63% No: 24% Same: 13%. (8 tasks)

Research Question II

Are pipes with smells **less understandable** than pipes without smells?

Correct: 80% vs. 67% on Non-smelly vs. Smelly pipes (2 tasks)

Refactorings for Pipes

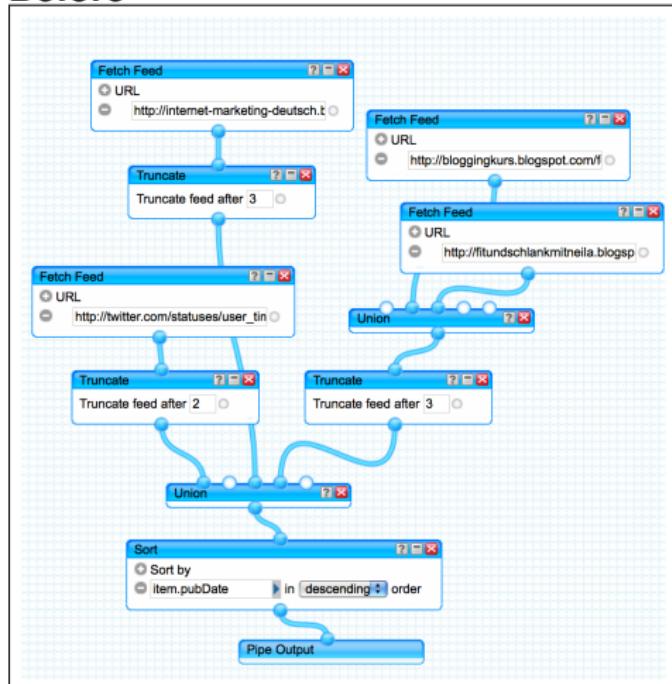
Targeted Smells	Refactoring Type	Refactoring
Laziness	Reductions	Clean Up Module Remove Non-Contributing Module Push Down Abstraction
Redundancy	Consolidations	Merge Redundant Modules Collapse Duplicate Paths
	Abstractions	Pull Up Module Extract Local Subpipe
Environmental	Deprecations	Replace Deprecated Modules Remove Invalid Sources
Population-Based	Population-Based	Normalize Operations Extract Global Subpipe

Consolidation: Collapse Duplicate Paths

Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Before



Consolidation: Collapse Duplicate Paths

Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Definition

P_{before} Joined generators

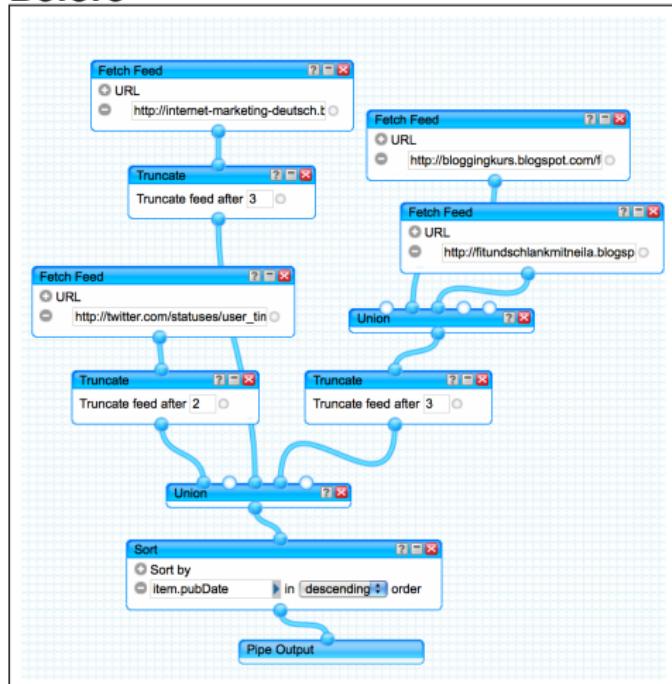
Params $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$
joined modules m, n

Transf. $\forall f \in n.\mathcal{F}$
move f to m
 $\exists w \in \mathcal{W} \mid out_wire(n, w)$
remove w

remove n

P_{after} $n, w \notin Pipe$
 $m'.\mathcal{F} = n.\mathcal{F} \cup m.\mathcal{F}$

Before



Consolidation: Collapse Duplicate Paths

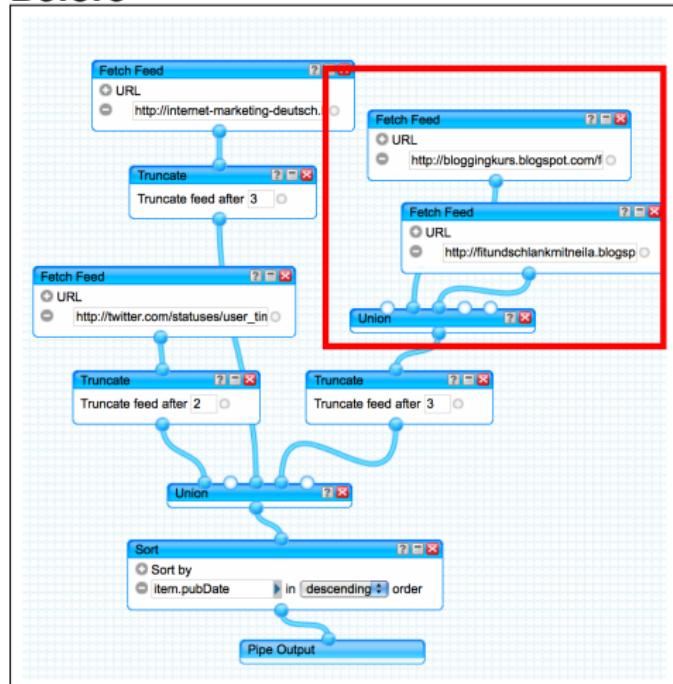
Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Definition

P_{before} Joined generators

Before



Consolidation: Collapse Duplicate Paths

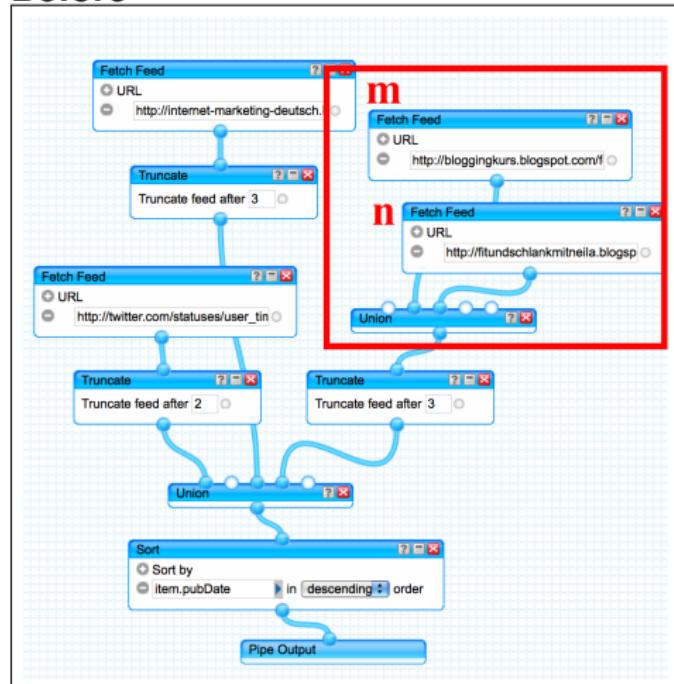
Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Definition

P_{before} Joined generators
Params $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$
joined modules m, n

Before



Consolidation: Collapse Duplicate Paths

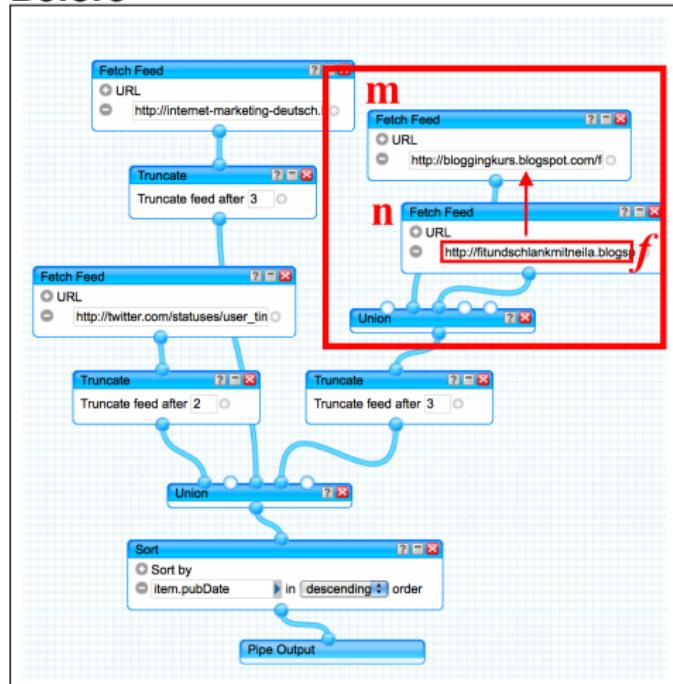
Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Definition

- P_{before}** Joined generators
- Params** $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$
joined modules m, n
- Transf.** $\forall f \in n.\mathcal{F}$
move f to m
 $\exists w \in \mathcal{W} \mid out_wire(n, w)$
remove w
remove n

Before



Consolidation: Collapse Duplicate Paths

Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Definition

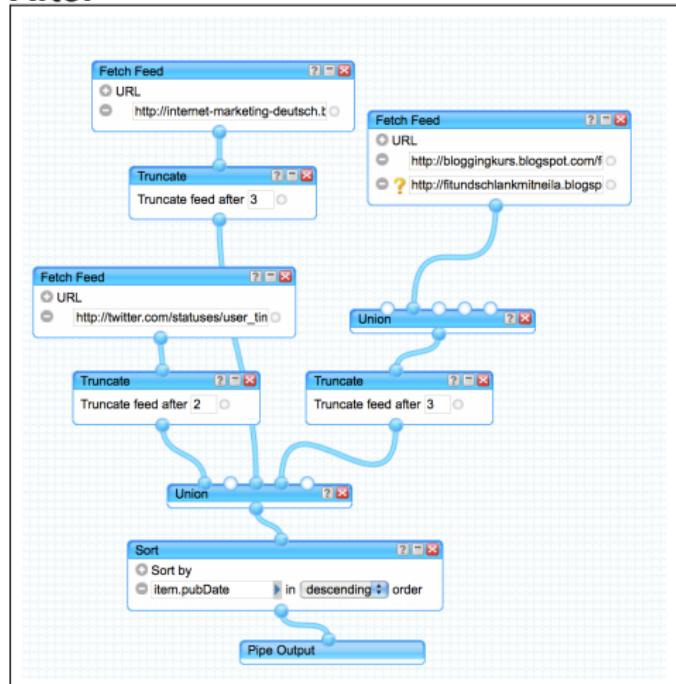
P_{before} Joined generators

Params $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$
joined modules m, n

Transf. $\forall f \in n.\mathcal{F}$
move f to m
 $\exists w \in \mathcal{W} \mid out_wire(n, w)$
remove w
remove n

P_{after}
 $n, w \notin Pipe$
 $m'.\mathcal{F} = n.\mathcal{F} \cup m.\mathcal{F}$

After

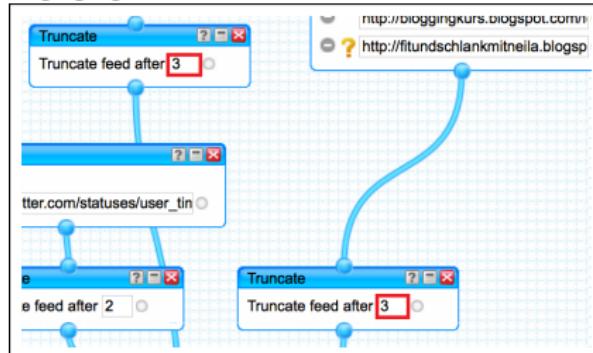


Abstraction: Pull Up Module

Pull Up Module

Pulls duplicate strings into new module

Before



Abstraction: Pull Up Module

Pull Up Module

Pulls duplicate strings into new module

Definition

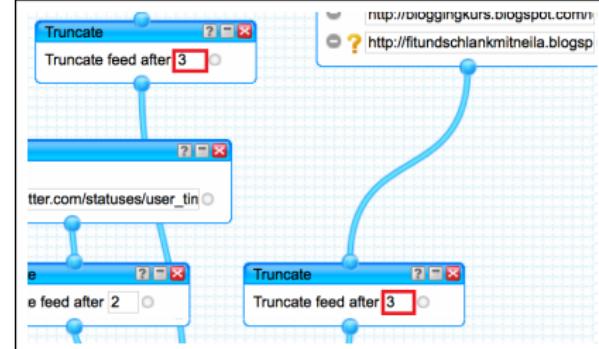
P_{before} Duplicate Strings
Params $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$,
fields f and g

Transf. add module m to $Pipe.\mathcal{M}$ |
 $\text{setter_str}(m)$
add field h to $m.\mathcal{F}$
set $h.value = f.value$
add wire w to $Pipe.\mathcal{W}$ |
 $\text{joined_fld}(m, f, w)$

add wire x to $Pipe.\mathcal{W}$ |
 $\text{joined_fld}(m, g, x)$

P_{after} $m, w, x, h \in Pipe$ |
 $h.value = f.value \wedge$
 $\text{joined_fld}(m, f, w) \wedge$
 $\text{joined_fld}(m, g, x)$

Before



Abstraction: Pull Up Module

Pull Up Module

Pulls duplicate strings into new module

Definition

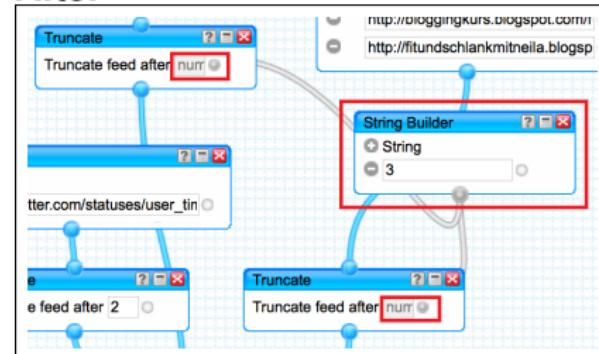
P_{before} Duplicate Strings
Params $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$,
fields f and g

Transf. add module m to $Pipe.\mathcal{M}$ |
 $\text{setter_str}(m)$
add field h to $m.\mathcal{F}$
set $h.value = f.value$
add wire w to $Pipe.\mathcal{W}$ |
 $\text{joined_fld}(m, f, w)$

add wire x to $Pipe.\mathcal{W}$ |
 $\text{joined_fld}(m, g, x)$

P_{after} $m, w, x, h \in Pipe$ |
 $h.value = f.value \wedge$
 $\text{joined_fld}(m, f, w) \wedge$
 $\text{joined_fld}(m, g, x)$

After

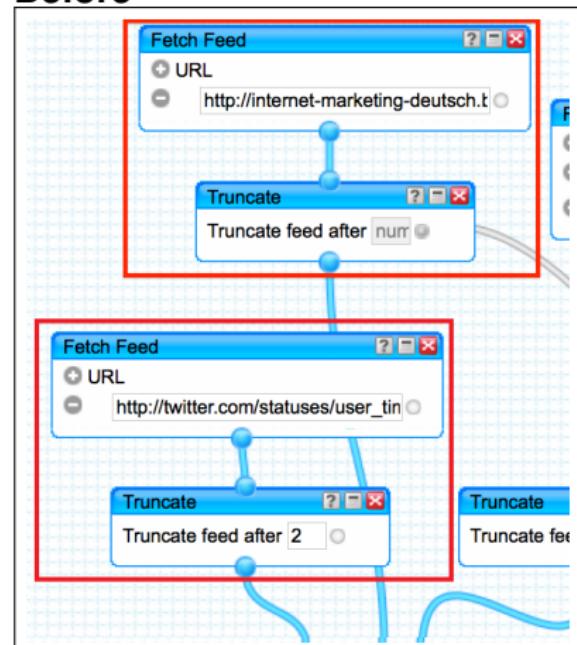


Population-Based: Extract Global Subpipe

Extract Global Subpipe

Replaces a path p with a subpipe if p is isomorphic to a path in the set of popular global paths extracted from the population.

Before



Population-Based: Extract Global Subpipe

Extract Global Subpipe

Replaces a path p with a subpipe if p is isomorphic to a path in the set of popular global paths extracted from the population.

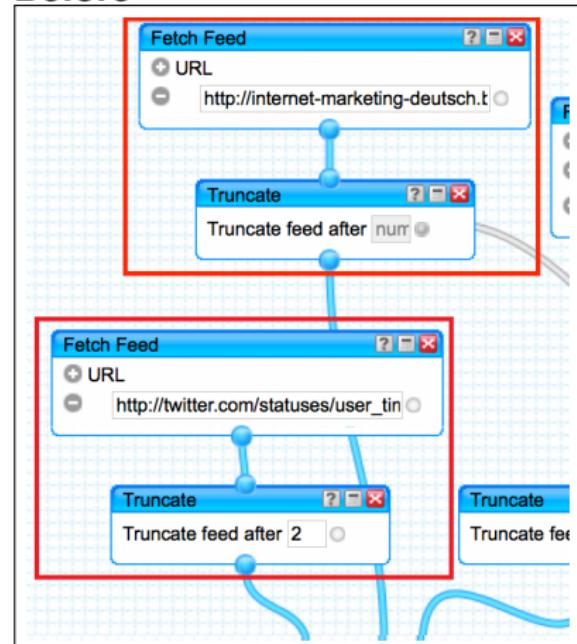
Definition

P_{before} Global Isomorphic Paths
Params $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$,
isomorphic Paths
Transf. for($p \in Paths$)

$subPipe = getSubPipe(p)$
connect $subPipe$ to $Pipe$
copy/rewire params from p
into $subPipe$
remove p

P_{after} $\forall p \in Paths \mid p \notin Pipes,$
 $\exists_1 getSubPipe(p) \in Pipe$

Before



Population-Based: Extract Global Subpipe

Extract Global Subpipe

Replaces a path p with a subpipe if p is isomorphic to a path in the set of popular global paths extracted from the population.

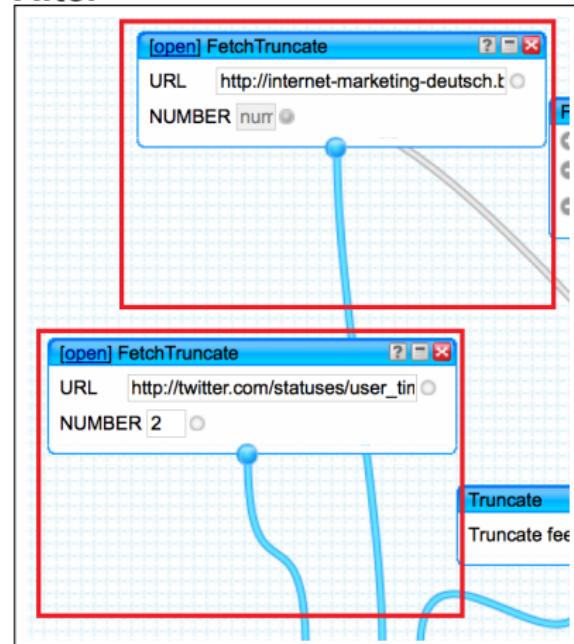
Definition

P_{before} Global Isomorphic Paths
Params $Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$,
isomorphic Paths

Transf. for($p \in Paths$)
 $subPipe = getSubPipe(p)$
 connect $subPipe$ to $Pipe$
 copy/rewire params from p
 into $subPipe$
 remove p

P_{after} $\forall p \in Paths \mid p \notin Pipes,$
 $\exists_1 getSubPipe(p) \in Pipe$

After



Study Design

Research Questions III

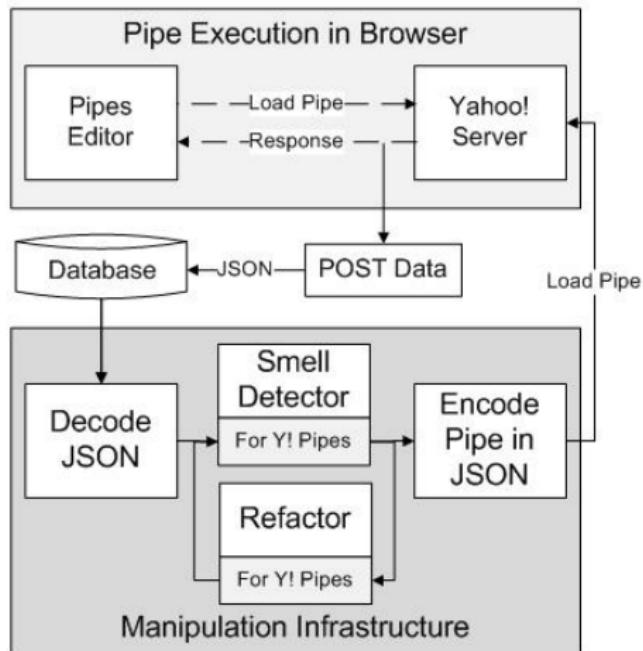
How often do the smells occur in the population of Yahoo! Pipes?

Research Questions IV

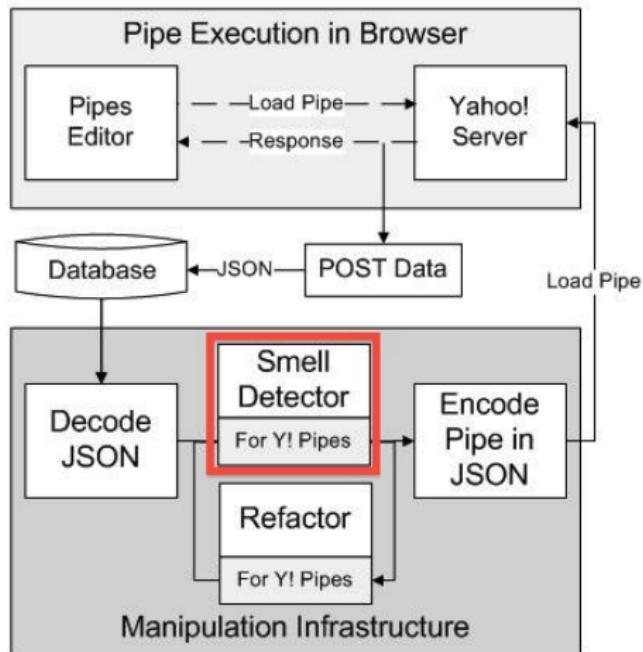
How effective are the refactorings at removing smells?

- Built infrastructure to detect smells and transform pipes
- Studied 8,051 Pipes developed by 5,957 authors

Study Infrastructure and Results

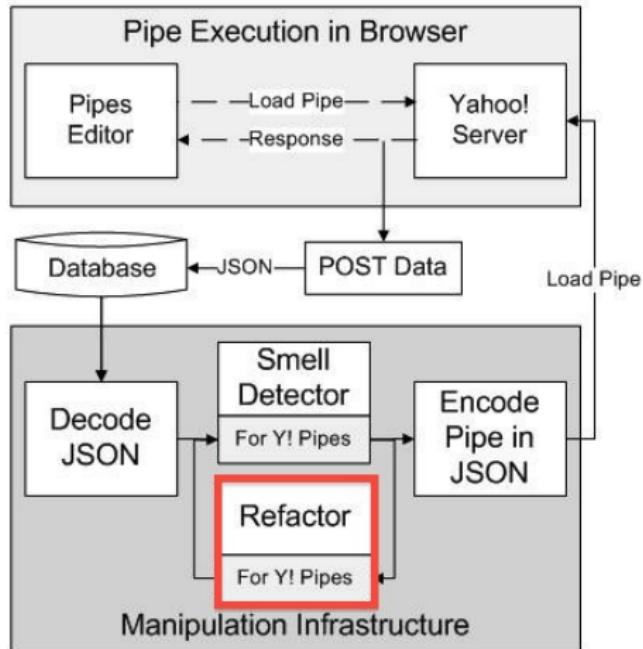


Study Infrastructure and Results



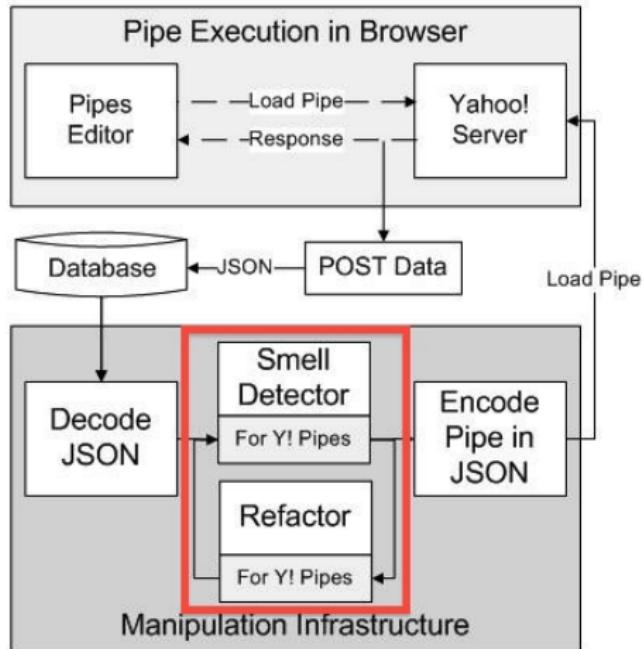
- 81% of 8,000+ pipes contains at least one smell instance

Study Infrastructure and Results



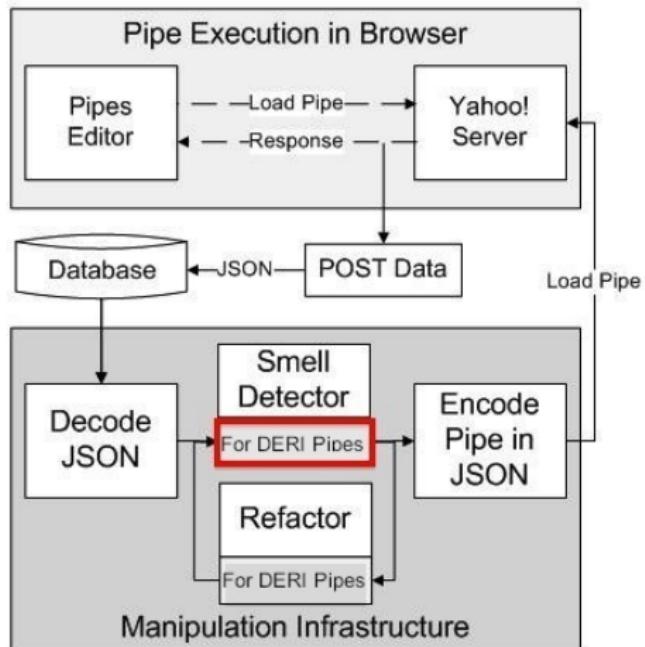
- 81% of 8,000+ pipes contains at least one smell instance
- 7 / 11 refactorings completely removed their targeted smells from the population

Study Infrastructure and Results



- 81% of 8,000+ pipes contains at least one smell instance
- 7 / 11 refactorings completely removed their targeted smells from the population
- Refactorings reduced the number of “smelly” pipes in the population from 81% to 16%

Study Infrastructure and Results



- 81% of 8,000+ pipes contains at least one smell instance
- 7 / 11 refactorings completely removed their targeted smells from the population
- Refactorings reduced the number of “smelly” pipes in the population from 81% to 16%
- Analysis of 77 DERI Pipes: 45% contain at least one smell

Summary

- 1 End users are creating mashups in increasing numbers
- 2 Mashups are often reused, yet suffer from many deficiencies
- 3 Smells are present in 81% of a sampling of 8,000+ pipes
- 4 End users generally prefer pipes that lack the smells
- 5 Refactoring are effective at removing smells
- 6 Definitions extend to other pipe-like mashup languages
- 7 Extends challenges and opportunities for refactoring

Future Work

- Evaluate if a refactoring tool would make development easier for end users
- Broaden the family of refactorings to target fault-proneness and to better utilize community information
- Adapt the refactorings to other domains of end user languages (e.g., Taverna, Scratch, Kodu, ...)
- Utilize community information to support end users earlier in the development lifecycle

This work was supported in part by NSF Graduate Research Fellowship CFDA#47.076, NSF Award #0915526, and AFOSR Award #9550-10-1-0406.

{kstolee, elbaum} @ cse.unl.edu

Potential Refactoring Tool

Smells

- Joined Generators (2)
- Unnecessary Module (1)
- Duplicate Strings (2)
- Isomorphic Paths (2)

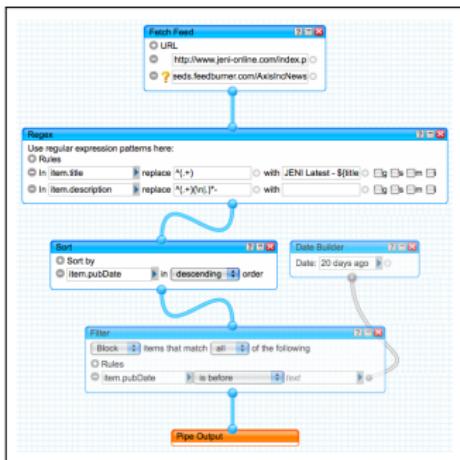
Refactor

Refactoring Examples

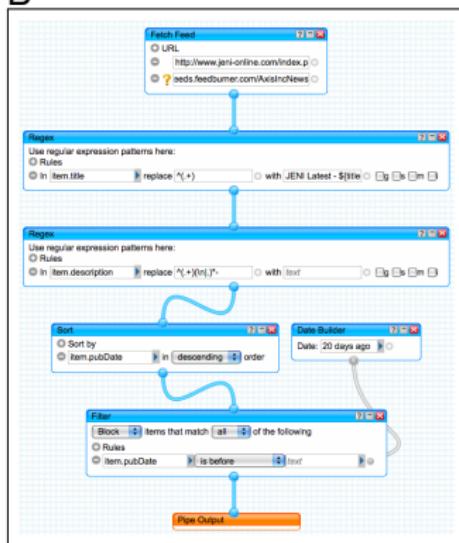
Joined Generators ▾

Example Task: Preference

A

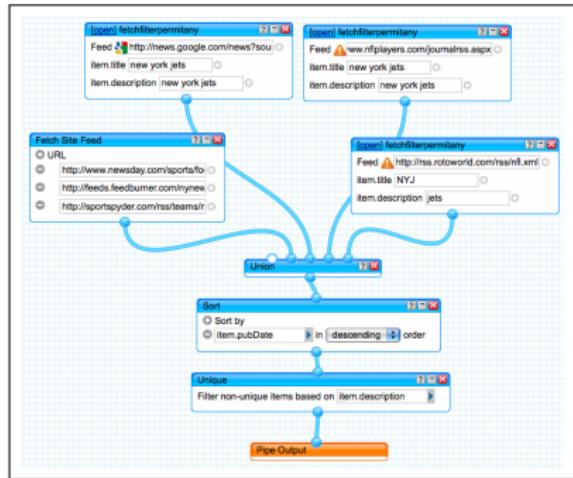


B



Question: Select the pipe that is easiest *to understand*

Example Task: Understanding



Select the pipe's output:

- 1 No Output
- 2 All of the content of the websites specified in the URL Builders.
- 3 The content of eight websites, filtered based on the presence of a user-defined value in the title of each item.
- 4 The content of four websites, filtered based on the presence of a user-defined value in the title of each item.

Refactoring Study Results

	Frequency	Max % Removed Individually	Max % Removed Collectively
Noisy Module	28%	18%	43%
Unnecessary Module	13%	100%	100%
Unnecessary Abstraction	12%	100%	100%
Duplicate Strings	32%	100%	100%
Duplicate Module	23%	72%	90%
Isomorphic Paths	7%	100%	100%
Deprecated Module	18%	100%	100%
Invalid Source	14%	99%	99%
Non-Conforming Op Order	19%	100%	100%
Global Isomorphic Paths	6%	100%	100%
All Smells	81%	—	80%

End User Study – Detailed Results (Preference)

Task	Smell	Subjects	% Smelly	% Non-Smelly	% Same
1	Joined Generator	10	10%	90%	0%
2	Duplicate String	10	10%	90%	0%
3	Redund. ops	10	20%	50%	30%
4	Invalid Src.	8	25%	38%	38%
5	Deprecated. Mod.	9	56%	33%	11%
6	Noisy Mod.	11	9%	73%	18%
7	Global Iso. Path	9	56%	33%	11%
8	Operator Order	8	13%	88%	0%

End User Study – Detailed Results (Understanding)

Task	Smell	Type	Subjects	% Correct	% Incorrect
9	Local Iso. Path	Clean	6	83%	17%
		Smelly	1	100%	0%
10	Joined Generators	Clean	4	75%	25%
		Smelly	5	60%	40%