Refactoring Pipe-like Web Mashups for End-User Programmers

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

May 25, 2011
End User Programmers

People who engage in programming activities to support their hobbies and work.
## End User Programmers

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

<table>
<thead>
<tr>
<th></th>
<th>Professionals</th>
<th>End Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number in U.S. (2005)</td>
<td>3 million</td>
<td>55 million</td>
</tr>
<tr>
<td>Typical Education</td>
<td>C.S. Degree</td>
<td>Other Degree</td>
</tr>
<tr>
<td>Role of Programming</td>
<td>It’s their job</td>
<td>It supports their job</td>
</tr>
</tbody>
</table>
Many Domains and Applications

Web Mashups:

Educational Games:

Spreadsheets:

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

- Smells matter to end users

- Mashup languages are hard for end users to understand
  Zang, N. and Rosson, M.B. Playing with information: How end users think about and integrate dynamic data. VL/HCC 2009

- Maintenance in mashups is common: nearly 24% of Yahoo! Pipes mashups were modified after they were made public
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)
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)
About Yahoo! Pipes
About Yahoo! Pipes
About Yahoo! Pipes
Motivating Example
Motivating Example
Motivating Example

[Diagram showing a network of nodes and arrows representing a feed processing pipeline with operations like fetching feeds from URLs, filtering non-unique items, unique filtering, sorting, and unioning results.]

- Fetch Feed
  - URL
    - http://rss.cnn.com/rss/money_mos
  - URL
    - http://feeds.wsjonline.com/wsj/xml
    - http://feeds.wsjonline.com/wsj/xml
- Fetch Feed
  - URL
    - http://rssfeeds.usatoday.com/Usat
    - http://feeds.sfgate.com/sfgate/rss/l
    - http://feeds.feedburner.com/inc/all
    - http://www.nytimes.com/services/xml
    - http://www.ft.com/rss/home/us
    - http://feeds.chicagotribune.com/ch
    - http://www.topix.net/rss/business/b
    - http://www.entrepreneur.com/feeds

- Unique
  - Filter non-unique items based on item.link
- Unique
  - Filter non-unique items based on item.link
- Unique
  - Filter non-unique items based on item.title

- Sort
  - Sort by
    - item.pubDate in descending order

- Union
- Pipe Output
Motivating Example

Diagram showing the process of fetching feeds, filtering non-unique items based on URL and title, and sorting items by publication date.
Motivating Example
Motivating Example
Motivating Example
Motivating Example
Motivating Example
Motivating Example
Motivating Example: Before and After

Before

- Redundant structures
- Broken data sources

After

- 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
Motivating Example: Before and After

**Before**
- Redundant structures
- Broken data sources

**After**
- 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

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.
Code Smell

Program characteristics that identify deficiencies in code
### Definitions

<table>
<thead>
<tr>
<th><strong>Code Smell</strong></th>
<th>Program characteristics that identify deficiencies in code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Refactoring</strong></td>
<td>A semantic preserving transformation on a program</td>
</tr>
</tbody>
</table>
## Definitions

<table>
<thead>
<tr>
<th>Code Smell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program characteristics that identify deficiencies in code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Refactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>A semantic preserving transformation on a program</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Smell Type</th>
<th>Smells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laziness</td>
<td>Noisy Module</td>
</tr>
<tr>
<td></td>
<td>Unnecessary Module</td>
</tr>
<tr>
<td></td>
<td>Unnecessary Abstraction</td>
</tr>
<tr>
<td>Redundancy</td>
<td>Duplicate Strings</td>
</tr>
<tr>
<td></td>
<td>Duplicate Modules</td>
</tr>
<tr>
<td></td>
<td>Isomorphc Paths</td>
</tr>
<tr>
<td>Environmental</td>
<td>Deprecated Module</td>
</tr>
<tr>
<td></td>
<td>Invalid Source</td>
</tr>
<tr>
<td>Population-Based</td>
<td>Non-Conforming Ordering</td>
</tr>
<tr>
<td></td>
<td>Global Isomorphc Paths</td>
</tr>
</tbody>
</table>
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
\[ \exists m, n \in \mathcal{M} \mid m \neq n \land \]
\[ m\text{.name} = n\text{.name} \land \]
\[ \text{gen}(m) \land \text{gen}(n) \land \]
\[ \text{connected_to_union}(m, n) \]
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**

\[ \exists m, n \in M \mid m \neq n \land \]
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**

\[ \exists m, n \in M \ | \ m \neq n \land m.name = n.name \land \text{gen}(m) \land \text{gen}(n) \land \text{connected to union}(m, n) \]
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**

\[
\exists m, n \in \mathcal{M} \mid m \neq n \land m.\text{name} = n.\text{name} \land \text{gen}(m) \land \text{gen}(n) \land \\
\text{connected to union}(m, n)
\]
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

\[ \exists m, n \in \mathcal{M} \mid m \neq n \land m . name = n . name \land \text{gen}(m) \land \text{gen}(n) \land \text{connected_to_union}(m, n) \]
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
\[ \exists m, n \in \mathcal{M} \mid m \neq n \land \\
\quad m.\text{name} = n.\text{name} \land \\
\quad \text{gen}(m) \land \text{gen}(n) \land \\
\quad \text{connected \_to \_union}(m, n) \]
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
\[
\exists f, g \in \mathcal{F} \mid f \neq g \land \\
\text{same_field_values}(f, g) \land \\
\text{owner}(f) \neq \text{owner}(g)
\]
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
\[ \exists f, g \in \mathcal{F} \mid f \neq g \land \text{same_field_values}(f, g) \land \text{owner}(f) \neq \text{owner}(g) \]
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

[Diagram of pipes populations]

Common Paths

Subpipes

Smelly Pipes
Population-Based Refactorings

Pipes Population  Common Paths

⇒

15 / 25
Population-Based Refactorings

Pipes Population  Common Paths  Subpipes

⇒  ⇒  ⇒
Population-Based Refactorings

Pipes Population  Common Paths  Subpipes  Smelly Pipes

⇒  ⇒  ⇒  ⇒
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
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 $\text{PGPaths}$, a pipe $P$ has this smell if:

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

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)
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?
## 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)
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

<table>
<thead>
<tr>
<th>Targeted Smells</th>
<th>Refactoring Type</th>
<th>Refactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laziness</td>
<td>Reductions</td>
<td>Clean Up Module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove Non-Contributing Module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Push Down Abstraction</td>
</tr>
<tr>
<td>Redundancy</td>
<td>Consolidations</td>
<td>Merge Redundant Modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Collapse Duplicate Paths</td>
</tr>
<tr>
<td>Abstractions</td>
<td></td>
<td>Pull Up Module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extract Local Subpipe</td>
</tr>
<tr>
<td>Environmental</td>
<td>Deprecations</td>
<td>Replace Deprecated Modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove Invalid Sources</td>
</tr>
<tr>
<td>Population-Based</td>
<td>Population-Based</td>
<td>Normalize Operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extract Global Subpipe</td>
</tr>
</tbody>
</table>
Consolidation: 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

\[ P_{\text{before}} \]

\[ P_{\text{after}} \]

\[ \text{Pipe} = (\mathcal{M}, \mathcal{W}, \mathcal{F}, \text{owner}) \]

\[ \forall f \in n.\mathcal{F} \]

move \( f \) to \( m \)

\[ \exists w \in \mathcal{W} \mid \text{out}_{\text{wire}}(n, w) \]

remove \( w \)

remove \( n \)

\[ n, w \notin \text{Pipe} \]

\[ m'.\mathcal{F} = n.\mathcal{F} \cup m.\mathcal{F} \]
Consolidation: Collapse Duplicate Paths

Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

**Definition**

\( P_{\text{before}} \)  Joined generators

Before
Consolidation: Collapse Duplicate Paths

Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Definition

\[ P_{\text{before}} \]

\[ \text{Params} \]

Joined generators

\[ \text{Pipe} = (M, W, F, \text{owner}) \]

joined modules \( m, n \)
Consolidation: Collapse Duplicate Paths

Collapse Duplicate Paths

Consolidate paths that are aggregated using the same module

Definition

\[ P_{\text{before}} \]

**P**arams

Joined generators

\[ \text{Pipe} = (M, W, F, \text{owner}) \]

joined modules \( m, n \)

\[ \forall f \in n.F \]

move \( f \) to \( m \)

\[ \exists w \in W \mid \text{out\_wire}(n, w) \]

remove \( w \)

remove \( n \)
Consolidation: Collapse Duplicate Paths

Collapse Duplicate Paths
Consolidate paths that are aggregated using the same module

Definition

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

\[ \forall f \in n.\mathcal{F} \]
\[ \text{move } f \text{ to } m \]
\[ \exists w \in \mathcal{W} \mid \text{out}_\text{wire}(n, w) \]
\[ \text{remove } w \]
\[ \text{remove } n \]
\[ n, w \notin \text{Pipe} \]
\[ m'.\mathcal{F} = n.\mathcal{F} \cup m.\mathcal{F} \]
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

\[ Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, \text{owner}), \]
fields \( f \) and \( g \)

\[ \text{Transf.} \]
add module \( m \) to \( Pipe.\mathcal{M} \mid \]
\[ \text{setter}_\text{str}(m) \]
add field \( h \) to \( m.\mathcal{F} \)
set \( h.\text{value} = f.\text{value} \)
add wire \( w \) to \( Pipe.\mathcal{W} \mid \]
\[ \text{joined}_\text{fld}(m, f, w) \]
add wire \( x \) to \( Pipe.\mathcal{W} \mid \]
\[ \text{joined}_\text{fld}(m, g, x) \]

\[ P_{after} \]
m, w, x, h \in Pipe \mid
\[ h.\text{value} = f.\text{value} \land \]
\[ \text{joined}_\text{fld}(m, f, w) \land \]
\[ \text{joined}_\text{fld}(m, g, x) \]
Abstraction: Pull Up Module

Pull Up Module
Pulls duplicate strings into new module

Definition

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

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

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

Transf.  
add module $m$ to $\text{Pipe}.\mathcal{M}$  
$\text{setter str}(m)$  
add field $h$ to $m.\mathcal{F}$  
set $h.\text{value} = f.\text{value}$  
add wire $w$ to $\text{Pipe}.\mathcal{W}$  
$\text{joined fld}(m, f, w)$  
add wire $x$ to $\text{Pipe}.\mathcal{W}$  
$\text{joined fld}(m, g, x)$

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.
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

$Pipe = (\mathcal{M}, \mathcal{W}, \mathcal{F}, owner)$, isomorphic Paths

$Transf.$ for ($p \in Paths$)

$subPipe = getSubPipe(p)$

connect $subPipe$ to $Pipe$

copy/rewire params from $p$ into $subPipe$

remove $p$

$P_{after}$

$\forall p \in Paths \mid p \notin Pipes, \exists_1 getSubPipe(p) \in Pipe$
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**

$P_{\text{before}}$ Global Isomorphic Paths

$\text{Pipe} = (\mathcal{M}, \mathcal{W}, \mathcal{F}, \text{owner})$, isomorphic Paths

$\text{Transf.}$ for $(p \in \text{Paths})$

$subPipe = \text{getSubPipe}(p)$

connect $subPipe$ to $\text{Pipe}$

copy/rewire params from $p$ into $subPipe$

remove $p$

$P_{\text{after}} \forall p \in \text{Paths} \mid p \notin \text{Pipes}$,

$\exists 1 \text{getSubPipe}(p) \in \text{Pipe}$
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

- 81% of 8,000+ pipes contain 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.
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
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%
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
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

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

Refactoring Examples

Refactor
Example Task: Preference

**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

<table>
<thead>
<tr>
<th>Smell</th>
<th>Frequency</th>
<th>Max % Removed</th>
<th>Individually</th>
<th>Collectively</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noisy Module</td>
<td>28%</td>
<td>18%</td>
<td>43%</td>
<td></td>
</tr>
<tr>
<td>Unnecessary Module</td>
<td>13%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Unnecessary Abstraction</td>
<td>12%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Duplicate Strings</td>
<td>32%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Duplicate Module</td>
<td>23%</td>
<td>72%</td>
<td>90%</td>
<td></td>
</tr>
<tr>
<td>Isomorphic Paths</td>
<td>7%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Deprecated Module</td>
<td>18%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Invalid Source</td>
<td>14%</td>
<td>99%</td>
<td>99%</td>
<td></td>
</tr>
<tr>
<td>Non-Conforming Op Order</td>
<td>19%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Global Isomorphic Paths</td>
<td>6%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td><strong>All Smells</strong></td>
<td><strong>81%</strong></td>
<td>—</td>
<td><strong>80%</strong></td>
<td></td>
</tr>
</tbody>
</table>
## End User Study – Detailed Results (Preference)

<table>
<thead>
<tr>
<th>Task</th>
<th>Smell</th>
<th>Subjects</th>
<th>% Smelly</th>
<th>% Non-Smelly</th>
<th>% Same</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joined Generator</td>
<td>10</td>
<td>10%</td>
<td>90%</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>Duplicate String</td>
<td>10</td>
<td>10%</td>
<td>90%</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>Redund. ops</td>
<td>10</td>
<td>20%</td>
<td>50%</td>
<td>30%</td>
</tr>
<tr>
<td>4</td>
<td>Invalid Src.</td>
<td>8</td>
<td>25%</td>
<td>38%</td>
<td>38%</td>
</tr>
<tr>
<td>5</td>
<td>Deprecated. Mod.</td>
<td>9</td>
<td>56%</td>
<td>33%</td>
<td>11%</td>
</tr>
<tr>
<td>6</td>
<td>Noisy Mod.</td>
<td>11</td>
<td>9%</td>
<td>73%</td>
<td>18%</td>
</tr>
<tr>
<td>7</td>
<td>Global Iso. Path</td>
<td>9</td>
<td>56%</td>
<td>33%</td>
<td>11%</td>
</tr>
<tr>
<td>8</td>
<td>Operator Order</td>
<td>8</td>
<td>13%</td>
<td>88%</td>
<td>0%</td>
</tr>
<tr>
<td>Task</td>
<td>Smell</td>
<td>Type</td>
<td>Subjects</td>
<td>% Correct</td>
<td>% Incorrect</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
<td>-------</td>
<td>----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>9</td>
<td>Local Iso. Path</td>
<td>Clean</td>
<td>6</td>
<td>83%</td>
<td>17%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smelly</td>
<td>1</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>10</td>
<td>Joined Generators</td>
<td>Clean</td>
<td>4</td>
<td>75%</td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smelly</td>
<td>5</td>
<td>60%</td>
<td>40%</td>
</tr>
</tbody>
</table>