Information and Software Technology 55 (2013) 1289–1303

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Discovering how end-user programmers and their communities use public repositories: A study on Yahoo! Pipes ☆

Kathryn T. Stolee *, Sebastian Elbaum, Anita Sarma

Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, NE, USA

ARTICLE INFO

Article history:
Available online 27 October 2012

Keywords:
End-user programmers
Community analysis
Artifact repositories
Web mashups
Diversity analysis

ABSTRACT

Context: End-user programmers are numerous, write software that matters to an increasingly large number of users, and face software engineering challenges that are similar to their professionals counterparts. Yet, we know little about how these end-user programmers create and share artifacts in repositories as part of a community.
Objective: This work aims to gain a better understanding of end-user programmer communities, the characteristics of artifacts in community repositories, and how authors evolve over time.
Method: An artifact-based analysis of 32,000 mashups from the Yahoo! Pipes repository was performed. The popularity, configurability, complexity, and diversity of the artifacts were measured. Additionally, for the most prolific authors, we explore their submission trends over time.
Results: Similar to other online communities, there is great deal of attrition but authors who persevere tend to improve over time, creating pipes that are more configurable, diverse, complex, and popular. We also discovered, however, that end-user programmers do not effectively reuse existing programs, submit pipes that are highly similar to others already in the repository, and in most cases do not have an awareness of the community or the richness of artifacts that exist in repositories.
Conclusion: There is a need for better end-user programmer support in several stages of the software lifecycle, including development, maintenance, search, and program understanding. Without such support, the community repositories will continue to be cluttered with highly-similar artifacts and authors may not be able to take full advantage of the community resources.

1. Introduction

The role of end-user programmers in communities is becoming more important as their numbers increase and they become more active contributors to artifact repositories. The population of end-user programmers is quickly over-whelming that of professional programmers, and was expected to have increased to 90 million in 2012 with 13 million describing themselves as programmers [2]. Despite their lack of a formal computer science education, end-user programmers are creating programs that are meaningful and have an impact not just on them or the businesses for which they work (e.g., a spreadsheet formula error reportedly cost a company millions of dollars [3]), but also on emerging online communities. These programmer communities are growing rapidly, exist in many domains, and facilitate knowledge sharing and code reuse. For example, the public repositories of mashups in several environments [4–7], web macros in CoScripter [8], web page modification scripts in Userscripts [9], and animations for several educational programming languages [10,11] have tens of thousands of program artifacts submitted by tens of thousands of users.

Professional programmers and end-user programmers confront many of the same challenges in interacting with their respective programmer communities. For example, as individuals, they need to configure sample code to run in their environments, use new APIs, or find a fault causing a failure. As a group, they need to learn how to build on, share, and contribute to the community. Studies on open source communities through the lens of public archives have been successful at unveiling some of the trends and needs in those communities (e.g., [12–15]). Yet, despite the growing importance of end-user communities, our understanding of the challenges, motivations, and needs of their communities is quite limited. Studies of online end-user communities have sought to characterize the participants roles using social evidence [16], but little is known about the type, quantity, and quality of artifacts contributed, and how end users and their contributions evolve over time.

☆ This paper is a revised and expanded version of a paper that appeared previously in the Proceedings of the International Symposium on Empirical Software Engineering and Measurement [1].

* Corresponding author.
E-mail addresses: kstolee@cse.unl.edu (K.T. Stolee), elbaum@cse.unl.edu (S. Elbaum), asarma@cse.unl.edu (A. Sarma).

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.10.004

This work aims to provide a better understanding of end-user programmers in a community setting. Our analysis targets the Yahoo! Pipes mashup language and community, which has been active since 2007. Programs have been created in this language by over 90,000 users, many who are active in collaborating with and helping one another through online forums [16]. Additionally, the community has built a public repository of over 100,000 artifacts. The popularity and availability of many diverse artifacts makes the repository amenable for analysis. We therefore perform a study of over 32,000 programs submitted to the Yahoo! Pipes public repository, characterizing the artifacts and using them to draw inferences about author behavior, the diversity of contributions, and community awareness. Specifically, we address three general research questions:

RQ1: *What are the characteristics of the Yahoo! Pipes community artifacts?*
RQ2: *What are the characteristics of the authors of Yahoo! Pipes programs?*
RQ3: *What are the characteristics of the most prolific authors?*

Our findings reveal several insights about end-user programmers in the Yahoo! Pipes community, including:

- Authors often submit pipes that are highly similar to other pipes in the repository; 60% of pipes in the repository have the same structure as other pipes, and 73% are within a distance of one from other pipes (where distance is measured in terms of additions/deletions/substitutions of modules and wires to obtain one pipe from another) (Section 5).
- There is great deal of attrition as over 81% of the authors we studied are active (i.e., contribute artifact(s) to the repository) for only 1 day (Section 6).
- Authors who persevere in the community tend to improve over time, creating pipes that are more configurable, diverse, complex, and popular. For example, the pipes submitted after an author has at least 1 month of experience in the community are significantly larger, more diverse, more popular, and more configurable (Section 6).
- The most prolific authors frequently create pipes that highly similar to pipes they created in the past. For example, 43% of the pipes created by the most prolific authors at structurally similar to pipes they created previously, a process we refer to as *tweaking* (Section 7).

A previous version of this paper explored the characteristics of community artifacts, authors, and prolific authors with respect to four metrics, popularity, configurability, size, and diversity for the Yahoo! Pipes community [1]. As an extension to that work, we refined the research questions, extended the description of the domain and discussion of the results, provide more details on the study set-up, and present an additional metric, distance, that is woven into the analyses for each research question. The distance metric captures the amount of changes needed to transform one pipe into another by counting the number of module or wire additions, deletions, or substitutions. This metric allows us to paint a clearer picture of the types of diversity that exists in the repository.

The rest of the paper is organized as follows. In Section 2, related work on end-user programmers and studies of online communities are discussed. Section 3 gives details on the Yahoo! Pipes programming environment, the particular subject of our investigation. In Section 4, we expand on the research questions and describe the study setup, including how we collected and analyzed the artifacts used in the study. The results to each of the three research questions are in Sections 5–7. The implications of the findings are discussed in Section 8 followed by threats to validity in Section 9 and conclusion in Section 10.

2. Related work

Three areas of related work require discussion: end-user programmers, studies on mashups, and studies on socio-technical communities with artifact repositories.

2.1. End-user Programmers

End-user programmers create programs and engage in programming activities to support their hobbies and work. What differentiates end-user programmers from professional programmers is that to end users, software is a means to an end, not the end itself [17]. These end users utilize programming environments and languages such as spreadsheets, databases, web macros, mashups, and many domain-specific languages, many of which have large public repositories (e.g., [4,8–10]).

Unlike professional programmers, end-user programmers do not have much support for all stages of the software lifecycle and may have a different lifecycle than that which is used by other types of programmers. Studying end-user programmers can reveal their needs, and researchers and practitioners have started applying software engineering techniques to provide support for end users' tasks. For example, version control has been introduced to help end users during development [5,18], debugging has been introduced to allow users to ask questions about output during development [19] or preview program output during testing [4], assertions have been used to increase the dependability of web macros during runtime [20], and strides have been made toward providing better program maintenance through refactoring support [21] and using program characteristics to predict how likely a program is to be reused [22]. However, software engineering support is far from pervasive in end-user programming environments.

Similar to professional programmers, many communities of end users contribute to central repositories of source code and artifacts. Repositories provide a mechanism for end-user programmers to share code and learn from the experiences of others, and tend to attract many participants to the communities. For example, Yahoo! Pipes has over 90,000 users [16], CoScripter has over 6000 users [22], Userscripts has over 57,000 users [9], and Scratch has over 500,000 users [23]. Beyond the number of participants, the repositories maintained by these communities contain thousands of public artifacts. For example, the Yahoo! Pipes repository contains over 100,00 artifacts [4], the CoScripter repository contains over 5430 public scripts [8], the Userscripts repository contains over 57,200 scripts [9], and the Scratch website contains over 47,800 galleries with as many as 1944 projects per gallery [10].

2.2. Mashups

Web mashup programming is among the most popular end-user programming domains. The research in this domain has diverged in two directions, one focusing on the mashup communities and the other on the mashup tools.

Many of the commercial mashup environments, such as Yahoo! Pipes [4], IBM Mashup Center'[5], xFruits [6], and JackBe [24], provide community resources like source code repositories and online help forums. Community analysis research on mashups has explored the conversations in discussion forums and looked at the social structure based on conversations [16].

A survey of mashup languages and features outlines many opportunities for additional development support [25], some of which we corroborate a need for in our findings (Section 8.2).

Other recent research in mashups analyzes the mashup tools themselves, proposing extensions for auto-completion or assisted composition [26–28], versioning support [18], refactoring [21], more domain-specific language support [29,30], or easier data integration from heterogeneous sources [31,32].

2.3. Studies on communities

Researchers in software engineering and computer supported cooperative-work have sought to understand the motivations and social organizations of developer communities. Research on communities with public artifact repositories has been particularly successful in open-source (e.g., [12,14]), and researchers are beginning to leverage repositories to also study end-user programmer communities (e.g., [16,23]). Even though the open-source and end-user programmer communities have many differences, there are commonalities in the activities performed by some roles in the open-source communities, such as bug fixers and bug reporters, and by programmers in end-user communities [33]. Here, we consider previous work that explores how developers join these communities and social factors that govern their contributions, as our study explores the characteristics of artifacts contributed by end-user programmers (Section 4).

Becoming an active member of an open source project is meritocratic, with joiners starting at low technical skill and low responsibility roles, such as participating in the mailing list. As they gain more experience learning both the technical parts and the social norms in the project, they advance to more central roles [12–15]. Contrastingly, becoming an active member in many end-user programmer communities seems to be universally accessible. Contributors are typically not required to demonstrate any expertise to participate, but this may be due, at least in part, to the fact that the projects tend to be smaller in scale and mostly individual [17].

In open source communities, most communication and project activities are archived through mailing lists, bug discussions, bug activities, and versioning systems [12]. End-user communities, on the other hand, have been observed to communicate through user comments associated with artifacts [22,23] and public message boards [16]. These differences in communication mechanisms may be rooted in fundamental differences between the groups, where generally the open-source programmers work toward a common goal and end-user programmers work toward an individual goal [17].

Social factors have been shown to be important in both the social organization and retention of developers in open-source communities. Studies have identified strong inherent social structures in the open-source community based on mail messages and found that successful members were also social hubs [34]. Power law relationships have been shown to hold on project sizes, the number of developers per project, and project memberships (number of projects joined by a developer). This is largely because of social relations, where members like to join projects that are already popular or join projects where they know some of the key players [35]. Another study found that the social network and the strength of the ties in the community was a good indicator for retention of members in the community in the face of external factors such as external projects and monetary incentives [36].

Yet for end-user communities, and specifically for Yahoo! Pipes – the particular subject of our study – the social factors may be different. Previous work has explored the nature of participation in the Yahoo! Pipes message boards [16], but little is known about the organization, participation, and growth patterns for the participants who contribute to the public artifact repository.

3. About mashups and Yahoo! Pipes

A mashup is an application that manipulates and composes existing data sources or functionality to create a new piece of data or service that can be plugged into a web page or integrated into an RSS feed aggregator. The Yahoo! Pipes environment, which we target in this work, provides language and development support for the creation of web mashups. One common type of mashup, for example, consists of grabbing data from some data sources (e.g., house sales, vote records, bike trails, map data), joining those data sets, filtering them according to a criterion, and plotting them on a map published at a site [37]. This type of behavior is naturally expressed in Yahoo! Pipes. Fig. 1a provides an example of the Pipes Editor, the Yahoo! Pipes development environment, and shows a pipe taken from the community that plots home sale information on a map or provides the output as a list (shown on the pipes information page in Fig. 1b).

The structure of a pipe resembles a graph, where the nodes are referred to as modules (boxes in Fig. 1a), and the edges are referred to as wires (connections between the modules). Each module has a name (e.g., *Fetch Feed*, *Filter*), and most modules contain fields that can hold hard-coded values or receive values via wire (e.g., the URL in a *Fetch Feed* module is a field, as are *Permit* and *item.description* in the *Filter* module). The data in a pipe flows in a directional manner from the top of the pipe through the output at the bottom. At the top is a module named *Fetch Feed*, which accesses external data sources and provides data to the pipe; this module contains two fields, each specifying a different website. The *Fetch Feed* module feeds data to a *Filter* module, which removes data from the feed based on the specified criteria. In the example, the *Filter* module permits data that matches any of the four specified criteria. Next, a *Location Extractor* module geotags the data, and last, the data flows to an *Output* module. The data that reaches the *Output* module is what appears when a pipe is executed, as shown in Fig. 1b.

An author can create a pipe from scratch or by cloning an existing pipe. Cloning can be performed by clicking the Save a Copy button in the Pipes Editor (Fig. 1a) or by clicking the Clone button on the pipes information page (Fig. 1b). Once a pipe has been created, it can be shared with the community; all pipe authors are free to contribute to the public repository. An author can commit any of their own pipes by clicking the Publish button from a pipes information page (Fig. 1b). Now, we describe the study that uses Yahoo! Pipes to address our research questions.

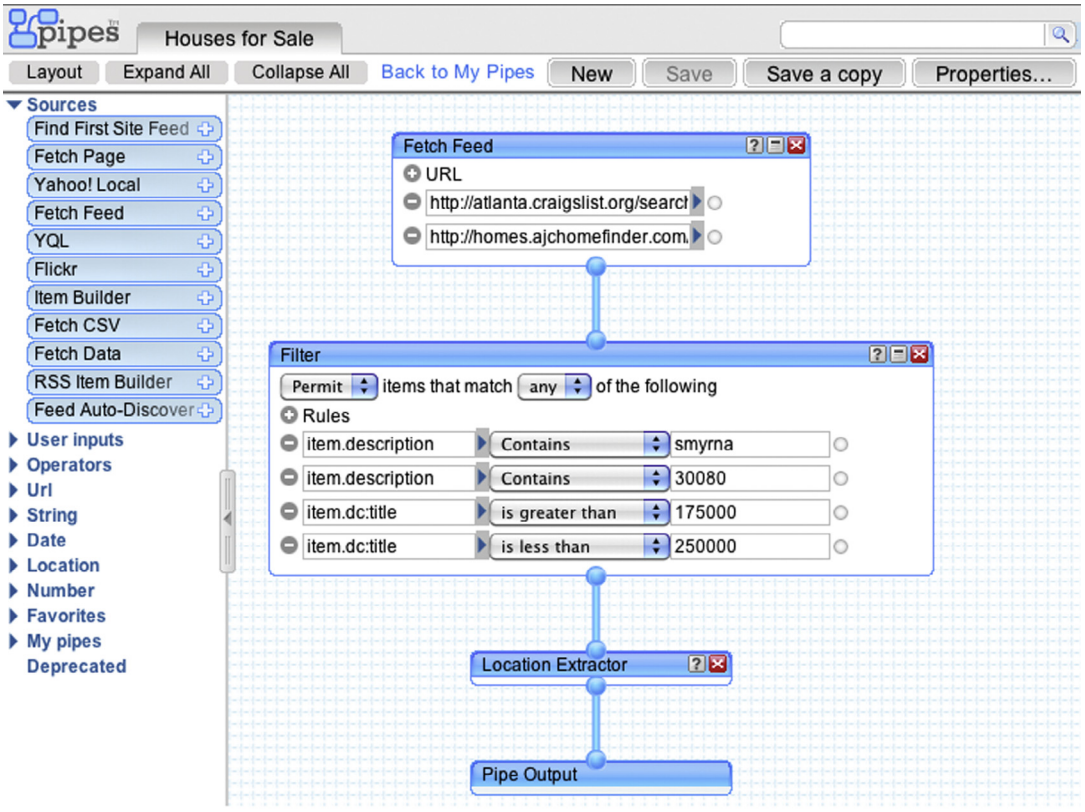
4. Study

Our broad research goal is to better understand *end-user programmer communities*. Through exploration of the Yahoo! Pipes repository, we have identified several research questions and conducted an empirical study to learn how communities and artifact repositories evolve, and uncover needs for end-user programmer support in several stages of the software lifecycle, including development, maintenance, search, and program understanding.

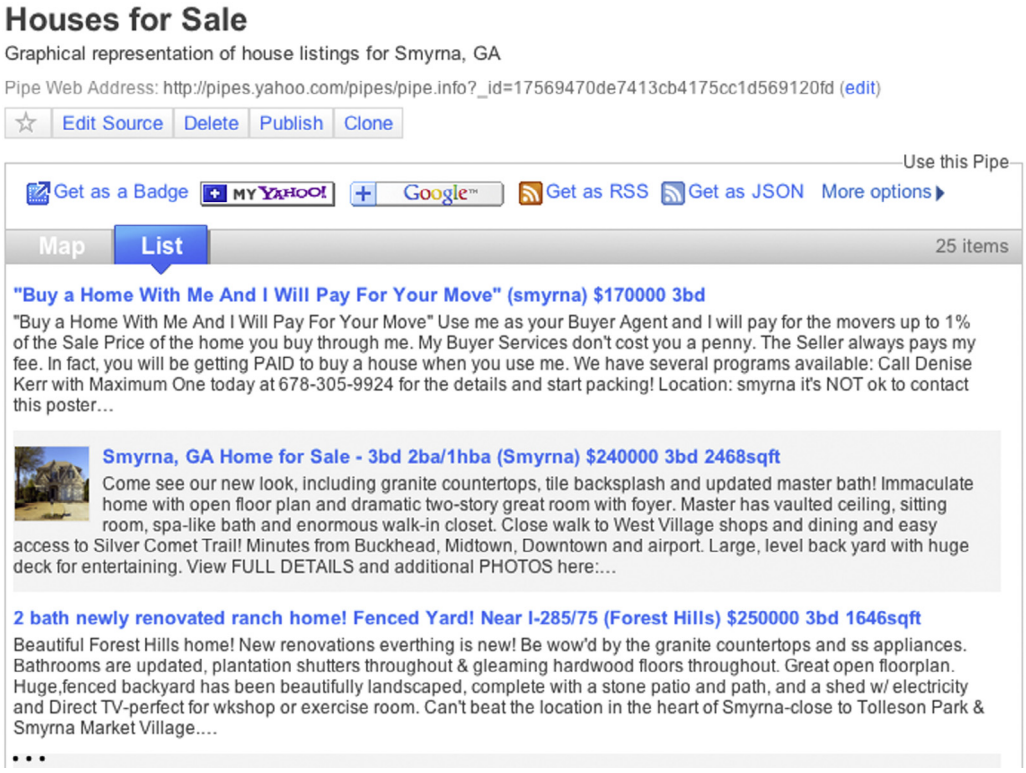
4.1. Research questions

We pose three broad research questions in this work. The first is about the artifacts in the repository, the second is about the authors and how pipe characteristics change as authors gain experience, and the third is about the characteristics of the most prolific (most contributing) authors in the Yahoo! Pipes community.

RQ1: What are the characteristics of the Yahoo! Pipes community artifacts considering structural diversity, distance to other pipes, popularity, size, and configurability?



(a) Pipes Editor



(b) Pipe Information Page

Fig. 1. Yahoo! Pipes environment.

RQ2: What are the characteristics of the authors of Yahoo! Pipes programs?

- *RQ2a:* How much attrition is there among the authors in the community?
- *RQ2b:* How much do authors typically contribute?
- *RQ2c:* What are the differences between pipes contributed when authors are new to the community versus when they have been involved for a determined amount of time?
- *RQ2d:* What are the differences between pipes contributed by authors with few contributions versus those contributed by authors with many contributions?

We view the most prolific authors as those who have the greatest impact on the repository in terms of quantity. This leads us to explore characteristics of these authors' contributions:

RQ3: What are the characteristics of the pipes created by the most prolific authors? That is, how different are a prolific author's contributed pipes compared to *their previous contributions* and *the pipes in the community*?

- *RQ3a:* What implications does the uniqueness of an author's contributions over time have for an author's evolution?
- *RQ3b:* What implications does the uniqueness of an author's contributions have for the types of activities in which the author engages?
- *RQ3c:* What implications does the uniqueness of an author's contributions have for the author's awareness of the community?

For our analysis we look at five dependent variables. Three are measured on the pipe in isolation: configurability, popularity, and size of the pipes. The other two, distance and diversity (or uniqueness), are measured when comparing a pipe to other pipes. In the previous version of this work [1], the diversity metric looked only at the structural similarity of pipes. However, it was possible to have two pipes with a high diversity level but where the only difference was a single module (e.g., the highest diversity level, 8, could describe two pipes that were different by just one module). In this work we introduce a complementary metric, distance, that captures the number of module changes (i.e., additions, deletions, or substitutions) required to obtain one pipes structure from another. Each of the dependent variables is defined in Section 4.2.2. We manipulate several independent variables related to author experience to uncover trends, including the days of experience an author had when the pipe was created and number of pipes created by an author.

4.2. Study setup

To address the research questions, we conduct an empirical study using artifacts from the Yahoo! Pipes repository. In performing this study, we had three main challenges: obtaining the artifacts, analyzing the artifacts, and measuring the differences (i.e., diversity and distance) among artifacts. In this section, we describe the methods for each of these steps.

4.2.1. Artifact collection

To perform this study, we had to obtain pipe representations from the Yahoo! Pipes repository. Yahoo! however, does not provide an API to obtain a pipe's structure outside their proprietary Pipes Editor. We used an infrastructure from a previous study to scrape pipes from the repository [21]. The scraping process works as follows: by executing searches on the Yahoo! Pipes repository, we obtained ids for those pipes returned by the search. For each id, we sent a *load pipe* request to Yahoo!'s servers; the response contained a JSON [38] representation of the Pipe. We stored the results in a database.

Between January and September 2010, we scraped 32,887 pipes from the Yahoo! Pipes repository that were created between February 2007 and September 2010. This number corresponds to the set of distinct pipes returned from approximately 50 queries against the repository, each of which returned a maximum of 1000 pipes. To obtain a representative pool of pipes without restricting the selection based on configuration or structure (since that may impact the effectiveness of this study in terms of measuring diversity among artifacts), we issued queries for pipes that utilized the 50 most popular data sources.[1]

4.2.2. Artifact analysis

Once the artifacts were collected, the next step was to analyze the pipe properties. We reused parts of the JSON decoding infrastructure from a previous study [21] to create an analyzable representation of each pipe, and then extended the infrastructure to measure the properties needed for this work. Each variable was selected to capture a concept, we use *size* to capture complexity, *configurability* to capture abstraction, *popularity* to capture artifact sharing and impact on the community, *diversity* to capture the uniqueness given various levels of abstraction, and *distance* to capture the amount of change needed to transform one pipe into another, given some level of abstraction (we consider topological changes). Here, we define each dependent variable measured for this study.

4.2.2.1. Size. Pipe size is measured in number of modules. In Fig. 1, the size of the pipe is four, since it has four modules. Every pipe is required to have an *Output* module, and so the minimum size is one for a pipe that has no behavior.

4.2.2.2. Configurability. The configurability of a pipe is measured by the number of *user–setter* modules in a pipe, where a *user–setter* module allows a user to specify field values at run-time [21]. For example, some pipes that search `Craigslist.com` allow the user to specify the item to search for each time the pipe is executed. That is, a user could use a configurable pipe to search for *couches* and then for *televisions* without changing the pipe structure or content. These run-time variables can be set using the pipe's information page (e.g., Fig. 1b). Configurable modules allow authors to create more general pipes that can serve a variety of purposes.

4.2.2.3. Popularity. The popularity of a pipe is measured in the number of clones; a clone is created when a user creates an exact copy of a pipe in the repository for their own purposes. This copy can then be saved and modified by the user, allowing them to reuse their own work or the work of others. The number of clones is reported for each pipe in the repository.

4.2.2.4. Diversity. Due to the size and the limited expressiveness of the Yahoo! Pipes language, we conjectured that there was much similarity among the artifacts in the repository. To assess this conjecture, we defined an ordinal diversity metric to measure the types of differences (i.e., uniqueness) among the artifacts and determine how much novelty exists in the repository as a whole. The diversity metric has eight levels (1–8) to describe differences between any two pipes in the repository, summarized in Table 1. Given two pipes, a low diversity level indicates that they are very similar (i.e., there are few differences between the pipes), whereas a high diversity level indicates the pipes are rather dissimilar.

[1] To facilitate replication, the data used in this analysis is available online: http://cse.unl.edu/kstolee/esem2011/artifacts.html.

Table 1
Summary of diversity metric levels.

Level	Criteria
1	The structure and content of the pipes are identical
2	The structure and number of fields per module are the same, but the field values relax
3	Topography/structure is the same; the field counts and values relax
4	Module bag (module names and counts) is the same
5	Module set is the same
6	Type bag is the same
7	Size is the same
8	The pipes exist

Level 1 represents pipes that have the same structure and content as another pipe in the population, possibly resulting from a clone. *Level 2* represents pipes with the same structure in terms of modules, the number of fields per module, and connections, but the field values can be different, whereas *Level 3* represents pipes with the same structure, relaxing all field values and counts. *Level 4* relaxes the connections between the modules but requires that the module bags[2] (module names and frequencies) are the same, and *Level 5* relaxes the frequencies and considers only the set of modules. *Level 6* considers the bag of modules based on types (i.e., *generator*, *setter*, *path-altering*, and *operator* [21]), *Level 7* considers only the number of modules, and *Level 8* is a catch-all for pipes not clustered in an earlier level (very unique pipes). The goal was to create a diversity gradient where the lower levels apply to pipes that are very similar, and the higher levels to pipes that are very diverse, with the assumption that differences in field values are less impactful than differences in topology. In summary, levels 1–3 consider changes to fields but keep the structures the same. Levels 4 and 5 consider changes to the connections between modules, but utilize the same language features (modules). Levels 6–8 represent pipes that are quite different.

4.2.2.5. Distance. The distance between two pipes can be measured by the number of additions, deletions, or substitutions required to get from one pipe to the other, using a modified Levenshtien's distance. This metric is different from the diversity metric; given a level of abstraction, we can find the *distance* between two pipes, which could indicate the number of changes needed to achieve one structure from another. Contrary to the diversity metric which systematically relaxes parts of the pipes to create clusters, the distance metric captures just the differences in topology.

For the sample we studied, we computed the minimum Levenshtien distance for each pipe, which represents the distance of that pipe to its closest neighbor. We considered only structural similarity, representing each pipe as a graph where the nodes are labeled according to the Pipes language (e.g., see the three example pipes in Fig. 2). This is equivalent to level 3 in the diversity metric, which retains just the pipe structure.

To compute the Levenshtien distance, each pipe structure is flattened to a string representation. The main structure of the pipe is treated like a parallel–serial graph, with parentheses and commas denoting parallel paths (e.g., `(x, y)` represents two parallel paths, `x` and `y`, separated by a comma) and spaces denoting serial paths (e.g., `xyz` is a serial sequence of `x – y – z`). For example, the pipe in Fig. 1a would be represented as `fetch filter locationExtractor output`. The string representations of each pipe in Fig. 2 appear below each pipe (with an added line break). The dashed lines in the bottom two pipes represent missing modules and/or wires, for ease of comparison. The distance from the top pipe to each of the lower pipes is one, which results from a missing wire (left side) or a missing module (right side). The distance between the bottom two pipes is two (because of the missing wire and module).

From Fig. 2, the reader may have noticed that not all pipes can be represented with a strict parallel–serial graph, as is the case with the top and right pipes. Modules with multiple output wires in which the wires go to modules at different levels in the graph will break the strict structure. For example, in the top graph, wires from `textinput0` go to `fetch` and to `filter`. If `fetch` and `filter` were parallel, this would be OK and could be represented as `textinput0 (fetch, filter)`. However, since they are serially connected, we lose precision when trying to force a parallel–serial representation. For example, `textinput0 (fetch, filter)` loses the wire between `fetch` and `filter`, and the representation `textinput0 fetch filter` loses the connecting wire between `textinput0` and `filter`. In the strict parallel–serial graph, the representation, `(textinput0, textinput0 fetch) filter` would capture all wires, but would imply that there are two `textinput0` modules, hence misrepresenting the modules in the graph. Thus, we must alter the representation of the parallel–serial graph to capture all wire connections.

All *user–setter* modules, such as `textinput0`, can have multiple output wires and be connected to nearly any other module in the pipe. To facilitate string comparisons that measure distance, the *user–setter* modules are treated as symbolic. That is, for each *user–setter* module, in order to capture all the connections of that *user–setter* module, we assign a symbolic value (i.e., `textinput0` and `textinputl` are symbolic names for these modules, which are both of type `textinput`), which is serially attached to each module to which it is connected. This allows us to represent each outgoing wire from the `textinput` modules while not duplicating the module.

Then, we permute the symbolic values within each type of *user–setter* modules, hence treating the modules as symbolic. It does not matter that `textinput0` is attached to `fetch` and `filter`, it only matters that some module with that same type (i.e., *textinput*) is attached to the `fetch` and `filter`. For example, the following are the two string representations for the top pipe in Fig. 2, and these are equivalent:

```
(textinput0, textinput0 fetch, textinputl) filter output
(textinputl, textinputl fetch, textinput0) filter output
```

When comparing two pipes, we identify the minimum distance among any of the string representations for the pipes (for pipes with no *user–setter* modules, there is only one string representation).

5. RQ1: Analysis of community artifacts

In this section, we address *RQ1* by exploring the characteristics of the contributed pipes considering each of the dependent variables: size, configurability, popularity, diversity, and distance. Generally, we observe a lot of cloning (high popularity) among the pipes, and that participants often submit pipes that are highly similar to other pipes in the repository.

5.1. Size

The average size across pipes in the community is 8.2 with a median of 6.0 modules per pipe. The distribution of sizes over pipes is shown in Table 2. We observe that more than two-third of the pipes have between three and ten modules, but that there is a long tail on the distribution where the largest pipe has 287

[2] A bag is an unordered collection that also contains a count on the number of times each item appears.

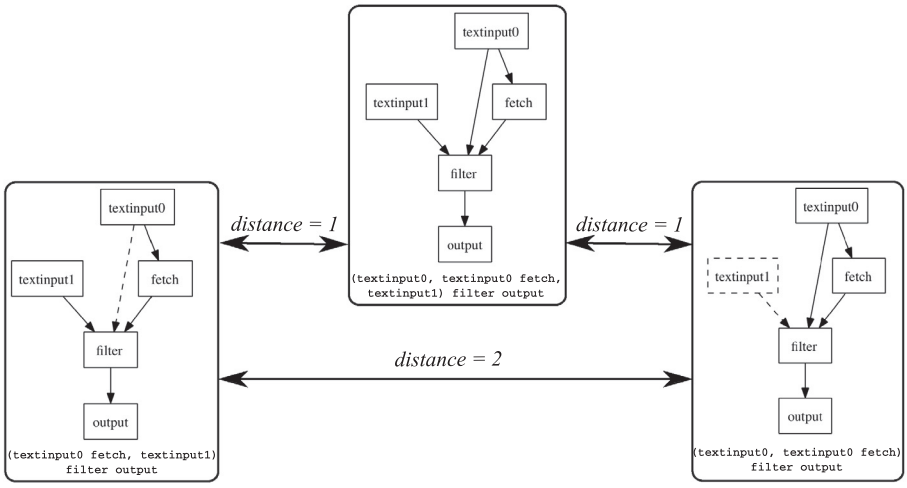


Fig. 2. Distance metric illustration.

Table 2
Size per pipe in community.

Modules	# Pipes	% Pipes (%)
[0,2]	2691	8.18
[3,5]	11,171	33.97
[6,10]	11,084	33.70
[11,20]	6510	19.79
[21,287]	1431	4.35

modules. This shows a large range in the complexity and size of pipes created by the community, indicating a range of skill levels and investment by the authors.

5.2. Configurability

The average number of *user–setter* modules across the pipes in the sample is 0.650, with a median of zero and a maximum of 73. Across all the pipes we studied, 33.81% have at least one configurable module, and an average of 20.76% of the modules in configurable pipes are *user–setter* modules. The majority of pipes were not made to be configurable, as shown in Table 3. There may be many reasons for this, such as a lack of understanding of the *user–setter* modules, being unaware of the benefits of generalizability in code, or being unable to configure some modules (e.g., some fields are set using a drop-down box, which cannot be configured at run-time).

5.3. Popularity

We associate a high number of clones with high popularity, as a clone represents an explicit decision to copy an existing pipe. Within the sample we studied, the average number of clones per pipe was 5.67 with a median of one clone per pipe with a maximum of 9180 clones. We observe that 17,874 (54.35%) of the pipes had been cloned at least once, and the distribution of clones over pipes is shown in Table 4. Approximately 11% of the pipes have more than five clones, so the overall majority have been cloned very few times. This low frequency of cloning may be because authors often cannot find pipes in the repository that suit their needs or that they cannot easily understand the behavior of pipes created by others.

Table 3
Configurability per pipe in community.

Configurable modules	# Pipes	% Pipes (%)
0	21,768	66.19
1	6301	19.15
2	2286	6.95
3	1590	4.83
[4,73]	942	2.86

Table 4
Popularity per pipe in community.

Clones	# Pipes	% Pipes (%)
0	15,013	45.65
1	7175	21.81
2	3290	10.00
[3,5]	3766	11.44
[6,10]	1632	4.96
[11,50]	1529	4.64
[51,9180]	482	1.46

5.4. Diversity

We create clusters among the pipes in the sample given the diversity metric in Table 1. When a pipe p matches another pipe at some levels l, we say that p is *clustered* at level l, where l is the minimum of all levels in which a match occurs. If we count the number of pipes that are clustered at level 1, we see that only 1731 (5.26%) of the pipes out of 32,887 have an exact match elsewhere in the sample, as shown in Table 5. The *Diversity Level* column indicates the level of diversity, the *# Clustered* column indicates the number of pipes that were clustered at the given

Table 5
Diversity of pipes in community.

Diversity	# Clustered	% of Pipes (%)
1	1731	5.26
2	15,186	46.18
3	19,319	58.74
4	20,262	61.61
5	24,316	73.94
6	29,346	89.24
7	32,862	99.93
8	32,887	100.00

Table 6
Minimum Levenshtien distance per pipe in community.

Distance	# Pipes	% Pipes (%)	Cumulative (%)
0	20,027	60.89	60.89
1	3976	12.09	72.98
2	2675	8.13	81.11
3	1728	5.25	86.36
4	1047	3.18	89.54
5	735	2.24	91.78
6	550	1.67	93.45
7	406	1.24	94.69
8+	1743	5.30	100.00

level, and the *% of Pipes* column identifies the percentage of pipes can be clustered at a given level.

Table 5 shows that there is much diversity among the pipes in the repository at low levels of abstraction (only 5% of the pipes are clustered at level 1), but not as much diversity at higher levels. At level 2 in which the field values are ignored, 46% have a match. Nearly 60% of the pipes have a match at level 3, and 89% at level 6. Similar to other repositories of code [39], the Yahoo! Pipes repository is full of duplication at higher levels of abstraction. This high frequency of similarity in structural abstractions may occur because authors can easily copy a pipe for their own usage by cloning and then change or add field values; there is little incentive to start from scratch if a user can start with a baseline pipe from another user. On the other hand, there is little value for the community to have a repository littered with duplicate programs. We investigate this further in Section 7.2.

5.5. Distance

We measure distance at a diversity level of three, which preserves the structure of the pipes but ignores all field information. The structure of a pipe defines the types of processing that are required to achieve the desired outcome, such as filtering, sorting, or joining lists. We use this level to determine the effort it would take to transform one pipe into another considering just the modules and connections.

The average minimum distance among all pipes in the sample we studied was 1.77, with a median of zero. Approximately 61% of the pipes have an identical match with distance zero,[3] as shown in Table 6. Nearly 95% of the pipes are within distance seven to another pipe in the population, which makes sense given the average pipe size is 8.2. Interestingly, over 80% of the pipe structures can be created by changing, adding, or deleting a maximum of two modules or wires from any given pipe.

6. RQ2: Analysis of community authors

In this section, we explore how much and how often authors contribute to the repository and the differences among pipes that have been created by authors with different levels of experience. We measure experience along two dimensions: the number of days of experience an author had when a pipe was created, and the total number of contributions by an author. We explore differences among the community artifacts by segmenting them along these lines. Generally, we observe that the most prolific authors create pipes that are larger, more popular, and more configurable than the less prolific authors, but the same cannot be said about the diversity and distance metrics.

6.1. RQ2a: Author attrition

From the sample of 32,887 pipes, we found they were created by 20,313 distinct authors. Most authors do not stay active in the community for very long, where activity is measured by the difference between the earliest and latest creation dates on the pipes they contributed. Approximately 82% of the authors were active for only 1 day, and only 13 authors were active for more than 3 years (maximum was 1253 days), as shown in Table 7. The *Duration* column indicates the length of time an author was active and the *# Authors* column indicates how many authors were active for this duration of time. As shown, the Yahoo! Pipes community suffers from attrition levels similar to other online communities [35].

6.2. RQ2b: Author contributions

Contributions are measured in number of pipes, and the average author contributes 1.62 pipes. Among the authors, 15,420 (76%) submitted only one pipe, as shown in Table 8. This accounts for 47% of the pipes in the sample. The remaining 24% of the authors are responsible for over 53% of the pipes, following a skewed distribution with a long tail; the most prolific author created 98 pipes.

6.3. RQ2c: Contributions based on experience (time)

Approximately 10% of authors submitted a pipe at least 1 month after submitting their first pipe (Table 7, sum of *% Authors* column from *1 Month* to *More than 3 years*). With this threshold in mind, we want to see if there are differences in the contributions made early in an author's experience (i.e., within the first month) versus late in their experience (i.e., after the first month). One month seemed reasonable time period for authors to gain sufficient experience with the environment, considering they may have developed other pipes before the first one we captured. For each pipe, the days of experience for the author when the pipe was created was measured; if the number of days was less than 31 (an

Table 7
Duration of author activity (days).

Duration	# Authors	% Authors (%)
1 day	16,592	81.68
2 days to 1 week	957	4.71
1 week to 1 month	655	3.22
1 month to 6 months	928	4.56
6 months to 1 year	537	2.64
1–3 years	631	3.10
More than 3 years	13	0.06
All	20,313	100.00

Table 8
Author contributions (in # of pipes).

Pipes contributed	# Authors	% Authors (%)	Total pipes	% of Total pipes (%)
1	15,420	75.91	15,420	46.89
2	2761	13.59	5522	16.79
3–5	1572	7.74	5595	17.01
6–10	381	1.87	2755	8.38
11–15	98	0.48	1211	3.68
16 or more	81	0.39	2384	7.25
All	20,313	100.00	32,887	100

[3] This is slightly higher than the number of pipes that were clustered at Level 3 in Table 5 because, for the distance analysis, we ignored structures that are not connected to main pipe – this adds approximately 700 (~2%) pipes to the distance level zero.

Table 9
Characteristics of pipes contributed early or late in an author's lifespan in the community. $\alpha = 0.01$.

Characteristic	Early (1)	Late (2)	H_0:	p-Value
# of Pipes	27,555	5332		
Diversity	3.519	4.126	$\mu_1 > \mu_2$	2.200×10^{-16}
Popularity	4.984	9.254	$\mu_1 > \mu_2$	2.200×10^{-16}
Configurability	0.614	0.838	$\mu_1 > \mu_2$	2.200×10^{-16}
Size	7.919	9.587	$\mu_1 > \mu_2$	2.200×10^{-16}
Distance	1.640	2.439	$\mu_1 > \mu_2$	2.200×10^{-16}

Table 10
Characteristics of pipes by authors with substantial contributions and authors with occasional contributions. $\alpha = 0.01$.

Characteristic	Occasional (1)	Substantial (2)	H_0:	p-Value
# of Pipes	30,503	2384		
Diversity	3.639	3.355	$\mu_1 > \mu_2$	1.000
Popularity	4.302	23.250	$\mu_1 > \mu_2$	2.200×10^{-16}
Configurability	0.644	0.729	$\mu_1 > \mu_2$	2.114×10^{-11}
Size	8.194	8.136	$\mu_1 > \mu_2$	0.001799
Distance	1.789	1.516	$\mu_1 > \mu_2$	0.9941

approximation used to represent 1 month), then the pipe was added to the early pool; else it was added to the late pool. The differences among the pools for the dependent variables are shown in Table 9. That is, within each pool of pipes, we averaged the values of the dependent variables across all pipes in the pool and report those averages. For example, the average diversity among all pipes in the early pool is 3.519, whereas the average diversity among all pipes in the late pool is 4.126.

For each dependent variable, (diversity, distance, popularity, configurability, and size), one-tailed Mann–Whitney tests[4] H_0:$\mu_{early} > \mu_{late}$ and $\alpha = 0.01$ reveal significant differences between the sample means. We therefore reject the null hypothesis; the sample means for all dependent variables are smaller for the pipes submitted within the first month versus after the first month of author experience. In other words, the diversity of the pipes in the early pool is significantly lower than the diversity of the pipes in the late pool, and this observation holds for all dependent variables. Thus, experience seems to play a role in increased diversity, distance, popularity, configurability and size of contributed pipes.

6.4. RQ2d: Comparisons based on contribution levels

In Table 8, we see that less than 0.5% of the authors created more than 15 pipes in the sampling of the repository. With this threshold in mind, we segment the pipes into two groups, those created by prolific authors who contributed more than 15 pipes, and those created by less prolific authors. For each author, the number of pipes they created was counted. If the author created more than 15 pipes, all their pipes were added to the *Substantial* pool, else their pipes were added to the *Occasional* pool. For each dependent variable, we report the average among all pipes in the *Occasional* and *Substantial* pools in Table 10, as was done with Table 9 for the *Early* and *Late* pools.

For three of the dependent variables, popularity, configurability, and size, one-tailed Mann–Whitney tests where H_0:$\mu_{occasional} > \mu_{substantial}$ and $\alpha = 0.01$ reveal significant differences.[5] Thus, we reject the null hypotheses; the pipes created by more prolific authors have more clones, are more configurable, and are larger. Note that for size we reject the null hypothesis even though the means support it; after further inspection we confirmed that this is correct as the mean numbers were caused by a handful of pipes in the *Occasional* group with more than 200 modules that account for its large mean value. For the diversity and distance metrics, the null hypothesis is *not* rejected (though, it is rejected for distance at a contribution level of five with $p = 4.569 \times 10^{-06}$ but not at a contribution level of ten; it is not rejected for diversity at either level). This is likely because, within the most prolific authors, some only submit pipes that are very similar to others they have submitted in the past, a phenomenon we explore further in Section 7.

[4] The data are ordinal and follow a skewed (non-normal) distribution, which motivates the use of the non-parametric Mann–Whitney test.

[5] We performed a sensitivity analysis by changing the contribution threshold to ten and five pipes per author, and found the same results at the same α value.

7. RQ3: Analysis of the most prolific authors

We view the most prolific authors as those who have the greatest impact on the repository in terms of quantity. This leads us to explore characteristics of these authors contributions. In this analysis, we concentrate on the individual authors and the uniqueness of their contributions, addressing each subpart of *RQ3*. To identify the most prolific authors, we selected authors who had contributed more than 15 artifacts to the repository. This threshold balanced our need to do individual author analysis while having enough samples to generalize across prolific authors. In total, we studied 81 authors (<0.5% of the authors in the study), who contributed 2384 pipes (~7% of the pipes in the study).

As with any community, there are many different types of authors who have many different characteristics. We use the uniqueness of the author contributions to explore three categories of interest for characterizing the participants and their contributions: author activities, author evolution, and author awareness.

7.1. RQ3a: Author evolution

As authors gain more experience with the Yahoo! Pipes language, we expected that they would more regularly create unique pipes and provide more value to the community. Here, we explore the implications of the uniqueness of pipe contributions over time to the authors evolution.

To investigate this conjecture, we perform two analyses. The first estimates a diversity and distance level of the author based on their ability to regularly create unique pipes compared to their previous contributions. The second measures the value of author contributions by correlating experience in terms of days with the uniqueness of their pipes compared to other pipes in the community, with the assumption that more unique contributions are more valuable to the community.

7.1.1. Assigning diversity and distance levels to authors

To assign a diversity and distance level to each author, we first perform a rolling analysis over time of the pipes contributed per author. That is, we identify the diversity and distance level as it is added to the set of pipes created by an author. The process for the rolling analysis is illustrated in Fig. 3. For each author, their

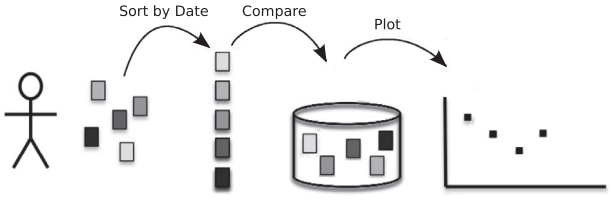
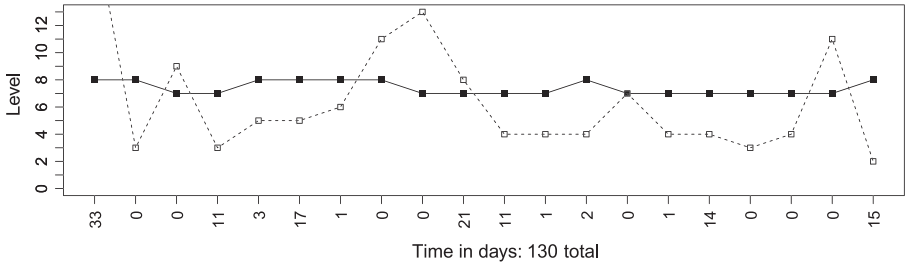


Fig. 3. Rolling analysis process.

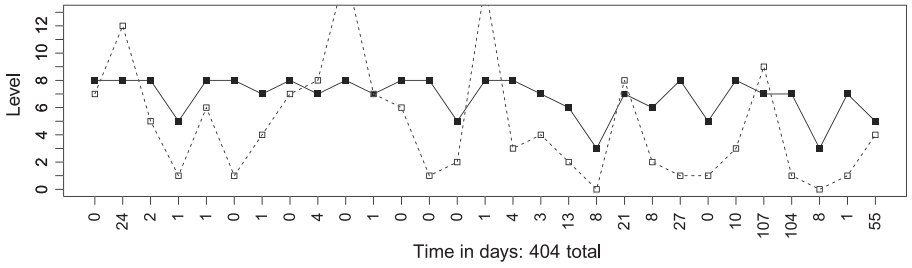
pipes are sorted by date. To begin, the earliest pipe is placed in a bin. Next, the second most-recent pipe is compared, pairwise, to each pipe in the bin with respect to diversity and distance. The minimum value for each metric is selected, as this represents the closest match to the pipe. These values are plotted on a graph and the pipe is added to the bin. This process continues for the third most-recent pipe and so forth, until all pipes have been processed. Note that diversity and distance build different graphs.

The end product is a graph, like that shown in Fig. 4 for the diversity metric (solid line) and the distance metric (dashed line). Time on the *x*-axis represents the number of days since the most recent pipe was submitted, and the diversity and distance levels are on the *y*-axis (note that the diversity metric has a maximum value of eight). More concretely, when considering only the diversity metric, represented by the solid line, the left-most dot represents the diversity level comparing the second pipe to the first. The second left-most dot represents the diversity level of the third pipe compared to the first two pipes. Thus, each subsequent pipe is compared to all those that came before it. For example, in Fig. 4a, we see the first dot at diversity level 8, with an *x*-axis label of 33. This means that when the second pipe was created, it had a diversity level of 8 compared to the first pipe and was created 33 days later. The second dot at diversity level 8 has an *x*-axis label of 0; it was compared to the two pipes that came before it and was created on the same day as the second pipe.

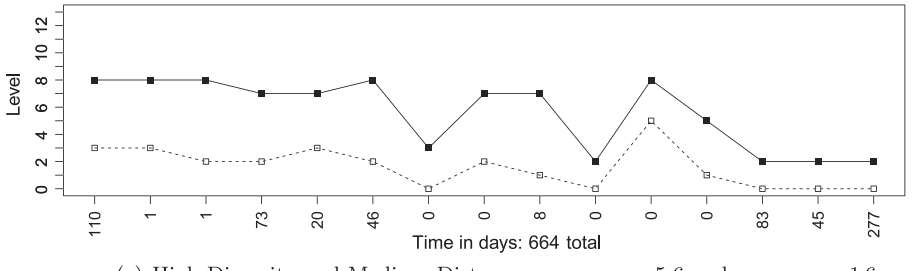
We calculate the average diversity and distance levels for each author to represent the average uniqueness of each pipe an author submitted to the repository, when compared to what they had previously submitted. A high diversity average, as in Fig. 4a–c, or a high distance in Fig. 4a and b, indicates an author who regularly



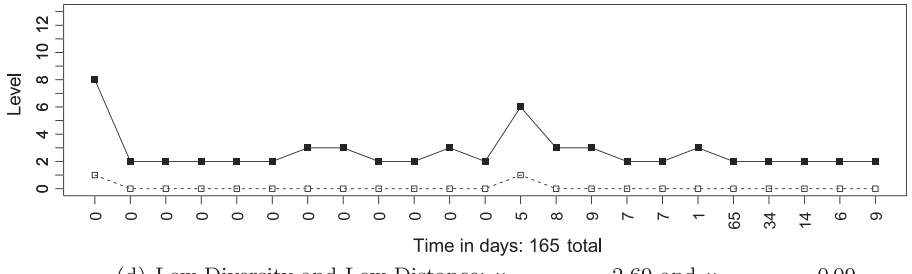
(a) High Diversity and High Distance; $\mu_{diversity} = 7.40$ and $\mu_{distance} = 6.45$



(b) High Diversity and High Distance; $\mu_{diversity} = 6.83$ and $\mu_{distance} = 4.72$



(c) High Diversity and Medium Distance; $\mu_{diversity} = 5.6$ and $\mu_{distance} = 1.6$



(d) Low Diversity and Low Distance; $\mu_{diversity} = 2.69$ and $\mu_{distance} = 0.09$

Fig. 4. Rolling analysis examples. The *y*-axis shows levels of diversity (solid lines) or distance (dashed lines). The *x*-axis represents the number days since the creation of the previous pipe.

Table 11
Author matrix for diversity and distance metrics by author.

	Distance			Sum
	Low [0,1]	Med. (1,3]	High (3,178]	
Diversity				
Low [1,3]	21	1	0	22
Medium (3,5]	4	9	1	14
High (5,8]	0	9	36	45
Sum	25	19	37	81

submitted distinct pipes. A low average, as is the case with the distance metric shown in Fig. 4d, indicates an author who regularly submitted pipes very similar to those they submitted previously.

The diversity metric is effective at detecting small changes in a pipe, such as changing field values (level 2) or adding/removing fields (level 3), as was the case for most of the pipes in Fig. 4d. This variation is not captured by the distance metric, which generally showed a distance value of zero. On the other hand, the diversity metric has a tendency to over-approximate the real diversity among pipes when the changes are from a small number of modules. For Fig. 4c, while the diversity metric measured the distances for the first six points at a high level, all of these pipes are within two or three steps of another, evidenced by the distance metric. In Fig. 4b, there are several cases in which a pipe that matched at diversity level 8 was also matched at distance level 1. There, the diversity metric over-estimates the real pipe diversity, but it is captured by the distance metric.

Table 11 shows the number of authors that fall into each of the diversity and distance levels, including the thresholds for classification. The diversity levels are based on the changes that are allowed within each range of levels. Diversity levels [1,3] allow for changes to the fields but preserve the structures, whereas levels (3,5] preserve the language features being used (i.e., the specific modules), but relax the structure. The highest diversity levels, (5,8] preserve very little information in the pipe. For the distance metric, [0,1] describes pipes in which at most one change (i.e., addition, deletion, or substitution) is needed between two pipes. The medium level, (1,3], describes pipes that are between one and three changes away; the highest level describes pipes that require at least three changes before two pipes are identical.

By both the diversity and distance metrics, close to half of the authors have high distance and high diversity, submitting pipes that are distinct to the previous ones they submitted. At the other end of the spectrum, approximately one quarter of authors submit pipes with low distance and diversity levels. These authors tend to submit pipes that are very similar to other pipes they have submitted, in essence using the public repository as their own personal repository and contributing clutter. Nine of the authors with high diversity are classified as medium with respect to distance. An example of this type of author is shown in Fig. 4c. In this case, while the diversity levels might be high, relatively few changes are needed to fill the distance between each new pipe and its closest match from the same author.

7.1.2. Assigning value to author contributions

We assume that more unique pipes are more valuable to the community and investigate if prolific authors with more experience create more valuable pipes. To do this, we measure the number of days of experience the author had when each pipe was created, and correlate that with diversity and distance against the community. For diversity and distance, there is a positive correlation with experience (Spearmans $r = 0.42136$ for diversity and $r = 0.38175$ for distance), so it appears that as the most prolific authors gain experience, the pipes they create tend to be more unique. This could indicate that over time, their contributions to the community become more valuable.

7.2. RQ3b: Author activities

We want to identify different practices that users demonstrate when creating and submitting pipes. We suspected that different authors followed different patterns in their submissions, where some would create consistently similar pipes and others would create consistently unique pipes. Each pipe submitted by an author represents an activity the author is performing. The uniqueness of one pipe compared to those created previously by that same author gives an indication of the goal the user had when creating the pipe. Using the rolling cluster analysis from *RQ3a*, we explore the implications that uniqueness (measured us- ing diversity and distance metrics) has for the types of activities in which the author engages.

On average, over 40% of the pipes created by a prolific author are highly unique (diversity levels 7 or 8) compared to the authors previous contributions. Additionally, one-third of an authors pipes were highly similar (diversity levels 1 or 2) compared to the authors previous contributions, and 43% had a diversity level 3 or lower. Using the diversity metric, we observe that authors tend to submit pipes that are either very similar with changes just within the modules while maintaining the structure (diversity levels 1–3), or very different (diversity levels 6–8) to what they submitted in the past.

Considering the distance metric, nearly 27% of pipes have a distance of four or more. Approximately 45% of the pipes had a distance value of zero and an additional 11% had a distance value of one. This means that 56% of the pipes are within one step of a pipe created previously by the same author. This is a 13% increase compared to the pipes clustered at diversity three or lower, showing many of the more diverse pipes according to the diversity analysis are within one step of a very similar pipe using the distance metric. It might be that the diversity metric is overshooting the real pipe diversity because it is sensitive to small structural changes that are captured by the distance metric. The highest distance value for a pipe added was 178, but on average the distance added was 2.96 with a median of one.

It's also important to note that the diversity and distance values for the within-author analysis are strongly correlated (Spearman's $r = 0.63935$), but in some cases, authors with high diversity commit pipes with relatively low distance, which means the difference between the pipes is the addition or deletion of just a couple modules and wires. Based on this analysis, we were able to map these results to two typical author activities. First, project **initiation** refers to pipes that are drastically different from those pipes an author had created previously. It is likely that the newly submitted pipe has a different purpose or goal from the previous ones. Second, pipe **tweaking** refers to pipes that are quite similar to those pipes created previously. It is likely that the author created something very similar to a pipe they submitted in the past (e.g., an author fixes a fault by changing the filter criteria for the home search in Fig. 1, giving a diversity level of 2 or 3, depending on the change). We see that for the average author, 52% of the pipes created represent new initiatives, while 43% represent tweaks.

Looking a little closer at the tweaks, we find that in some cases the tweaks may be the result of a language limitation. For example, for the author depicted in Fig. 4d, 21 of the 23 pipes had the same topology of three modules: a *fetch* that retrieved a URL, a *translate* that translated the website to English, and an output module. The differences between the pipes were the values of the URLs and translation language. While it would be possible to parameterize the *fetch* module by connecting a *user–setter* to it, it is not possible to connect a *user–setter* module to the *translate* module because its

Table 12
Average submission awareness per pipe by prolific authors.

Awareness	Avg. % of pipes	
	Diversity (%)	Distance (%)
None	51.04	51.02
Local	20.35	38.77
Community	28.61	10.20

field value is set with a drop-down box. Thus, the limitation of the language prevents the user from creating one configurable pipe and forces them to create many instances of highly-similar pipes. In this case, one may ask why share it with the community. We discuss this in Section 8.

7.3. RQ3c: Author awareness

Authors demonstrate different levels of awareness about what they submit, evidenced by how unique a submission is to their own pipes compared to the submissions uniqueness to the public repository (the uniqueness compared to an authors own pipes will always be greater than or equal to the uniqueness compared to the community). Using the diversity and distance metrics, we study this awareness using the uniqueness of each pipe compared to *the author's previous contributions* (local) and *the pipes in the community* (community).

Using the high, medium, and low levels for diversity and distance shown in Table 11, we classify the uniqueness for each pipe as high, medium, or low. For each pipe contributed, we look at how the local uniqueness differs from the community uniqueness, and then draw conclusions about the awareness demonstrated by the author when submitting the pipe. This is done for each the diversity and the distance metrics separately.

We observe that authors submit some pipes that are very similar to other pipes they already submitted (i.e., low local uniqueness, which implies low community uniqueness), so we say these pipes demonstrate *no awareness*. Other pipes are very unique compared to what the author had done in the past, but very similar to other pipes in the community (i.e., local uniqueness is strictly greater than community uniqueness); these pipes show *local awareness*. Last, there are pipes that are very unique compared to what the author had done in the past and also very unique compared to the community (i.e., high local and community uniqueness, or medium local and community uniqueness); these pipes may demonstrate *community awareness* (we also recognize the possibility that the pipes could be coincidentally unique compared to the community).

Table 12 shows that 50% of the pipes submitted by the most prolific authors represent no awareness, using both the diversity and distance metrics. Using diversity, 20% of the pipes demonstrate local awareness, but with distance, nearly 40% demonstrate local awareness. In exploring the data further, there are many pipes (15%) with a high local diversity but low global diversity, perhaps resulting from a clone of another author's pipe. For community awareness, the diversity metric shows that nearly 30% of the pipes represent community awareness while only 10% of the pipes do using the distance metric. The differences between the metrics at the local and community levels likely appear because the diversity metric tends to over-approximate diversity at higher levels of abstraction, where some of these changes are captured by the distance metric.

8. Discussion

From the analysis, we have made several general observations about the Yahoo! Pipes community, and these have led to some implications on how to better support the community. We note that these observations are based on an analysis of a single artifact repository, and may not generalize beyond the scope of our study.

8.1. Observations

8.1.1. Few authors are responsible for most artifacts

Like with other online communities, Yahoo! Pipes suffers from attrition and the contributions of the participants follow a skewed distribution with a long tail. Most of the participants (>81%) are active for only 1 day, and only 24% of the participants are responsible for over 53% of the artifacts in the repository.

8.1.2. Authors evolve over time

Pipes created early in authors' careers are significantly less diverse, popular, configurable, large, and have shorter distances to pipes with similar structures. This is shown by the significant differences between the groups of pipes when controlling for experience (Table 9) and by the positive correlations between average diversity and distance levels with the community and days active in the community for the most prolific authors. Additionally, author diversity (Table 11) is strongly correlated with the total time an author is active in the community (Spearman's $r = 0.61111$). These are positive observations for the community, showing that authors are able to grow over time.

8.1.3. Most pipes are very similar at lower levels of abstraction

We find that approximately 60% of pipes are structurally similar to others in the repository (Level 3 in Table 5 and Level 0 in Table 6), indicating much duplication in the repository and potentially little community awareness among all authors. Additionally, 73% of the pipes have a minimum distance of one or less (Table 6), indicating that beyond the structural diversity, an additional 13% of pipes are within one step (addition, deletion, or substitution of one wire or module) of another pipe in the repository.

8.1.4. Authors have varying levels of awareness

The trend of community awareness among all authors is unclear, as few exact matches in the repository indicates high awareness (or just a lot of diversity in the repository), but little diversity at higher levels of abstraction indicates low awareness. The low frequency of exact clones found in the repository (~5%, Table 5) and the fact that nearly a quarter of the pipes are distance two or more away from the closest match in the repository (Table 6) would suggest that authors may have some understanding of community and the value of contributing pipes that are somewhat different.

About 43% of the pipes submitted by the most prolific authors represent tweaks of something they have created in the past and 50% of the pipes submitted by the most prolific authors have the same structure (if not also the same fields – resulting from diversity levels 1–3) as other pipes they had submitted in the past (Table 12). This suggests that even the most prolific authors do not have much awareness of the community.

Yet for a minority of the pipes submitted by the most prolific authors (30% by the diversity metric, or 10% by the distance metric), the pipes are very different from what the authors created in the past and different from what exists in the repository (Table 12). While this may represent community awareness, further study is needed to understand if the author coincidentally or intentionally created highly unique pipes.

8.1.5. Experienced authors make more configurable pipes

A third of all pipes across the community are configurable (Table 3), and authors with more experience (Table 9) and who create more pipes (Table 10) tend to make pipes that are significantly

more configurable. This indicates that authors with more experience may have a greater interest in serving themselves or the community through their more configurable contributions. It should also be considered that a lack of configurability might be a product of language limitations rather than a conscious decision by the programmer.

8.1.6. Community participants seem interested in utilizing the community knowledge

Most pipes have been cloned at least once, with a small minority of pipes having hundreds, if not thousands, of clones (Table 4). This indicates that many participants are likely interested in building on the expertise of others.

8.2. Implications

8.2.1. Authors may need artifact maintenance support

Approximately one-quarter of the prolific authors were seen to contribute pipes with low diversity or low distance (Table 11), and over 43% of the author submissions represented tweaks on already-submitted pipes. Additionally, approximately 30% of the prolific authors change, on average, at most one module per pipe they submit (Table 11). This indicates that authors may be using the public repository as a private repository to store incremental changes on pipes, and they may benefit from a versioning system, which has been identified as a desirable feature for mashup environments [25]. It may also be that authors are unable to configure their pipes or do not know about the configuration options, and so they are forced to create pipes with few deviations from existing pipes.

8.2.2. The repository may need moderators

We observed that pipes created by prolific authors are significantly more popular and configurable (Table 10). However, with the number of authors who are only active for a short period of time (Table 7), the repository gets cluttered with highly similar and less configurable pipes. An alternative approach might be to notify authors of highly similar, existing pipes, or to have some controls on the quality or uniqueness of pipes before they're shared publicly. Considering that 70% of the pipes submitted demonstrate no awareness or local awareness (Table 12), a mechanism to alert authors when they are re-inventing the wheel might reduce the clutter in the repository and support end users in becoming more efficient.

8.2.3. Authors may need better search for the repository

Only 5% of the pipes have an exact clone elsewhere in the community, but over 46% have an exact match except for the field values (Table 5). It may be that the high frequency of similarity is a result of an author's inability to find an appropriate pipe from the clutter in the repository. Current search mechanisms only allow authors to search by URL, modules used, tags, output format, or keyword, and there is no support for users to search by partial structures (if they know part of a solution), quality, popularity, configurability, or behavior; few mashup environments offer this kind of support for discoverability [25]. Further study is needed to measure the semantic similarity between syntactically similar pipes to determine if they solve the same problem.

8.2.4. Authors may need notification of changes in related pipes in the repository

Currently, if authors create pipes that are later refined and posted as new pipes (i.e., they were cloned, modified, and resubmitted), which may explain some of the low distance submissions by some of the most prolific authors, then any clients of the original pipe would not be aware of the change. An awareness of the ensuing changes (if the modification improved the pipe, for example, by fixing a fault or improving efficiency) will help the pipe clients to benefit from the changes. Similarly, if a user created a pipe that depended on a *subpipe*[6] but later that *subpipe* was modified or deleted, a lack of awareness of these changes could cause the client pipe to behave differently than originally intended or fail silently. Thus, some notification system to alert cloned pipes or users of *subpipes* about changes in the relevant pipe might be beneficial.

8.2.5. Authors may need help understanding pipe behavior

Since there is no correlation between popularity and uniqueness (Spearman's $r = 0.06088$ with the diversity metric, $r = 0.10973$ for the distance metric), it is likely that either other authors are unable to understand the more unique pipes, authors cannot find what they need in the repository, or that the highly unique pipes solve a very narrow problem. If the pipes cannot be understood, authors may re-invent the wheel and create their own pipe with different syntax yet the same semantics. Perhaps there needs to be better support for helping authors understand the semantics of pipes created by others. One way to go about this would be through the inclusion of a *comments* module so authors can annotate their code, or through automatically-generated documentation.

8.2.6. Authors may need better development support

Given that approximately 50% of the authors do not consistently produce unique pipes (Table 11), pipes are not very configurable (33%, Table 3), and most pipes contain unfavorable characteristics (as found in previous studies [21]), there is an implied need for better compositional support to allow authors to create higher quality, more diverse and general pipes. Considering also the high frequency of tweaking (43% of pipes) that is performed by the authors and the high frequency of pipes with similar structures (59% in Table 5, and 73% in Table 6), authors would likely benefit from support in composition and design. This support could guide users on how to make their pipes more configurable by adding *user–setter* modules, allow for integration of multiple pipes, or to allow them to create test cases to verify program behavior in a controlled setting (rather than relying on RSS feeds, which update frequently and can make testing difficult).

Recent work in community analysis has shown that Yahoo! Pipes contributors sometimes turn to online help forums when the development support is insufficient for their needs [16]. It is unknown how much of the community that contributes to the discussion forums also contributes to the artifact repository, and vice versa. It would be interesting to explore how these two groups overlap and the trends that might emerge, but such an analysis is outside the scope of this work, since we analyze only the contents of the artifact repository.

9. Threats to validity

Here we discuss the threats to validity for our study, which fall into three categories: external, construct, and internal.

9.1. External

In this study, we consider only one domain, that of Yahoo! Pipes. Our results may not generalize to other end-user repositories, but we attempted to structure our analysis in such a way that the new diversity and distance metrics and notions of popularity, size, and configurability could be defined for new domains and then the results compared to ours.

[6] A *subpipe* is a language feature that allows a pipe to be embedded in another pipe and appears as a single module where the structure is not visible.

The sample of pipes we scraped are those that were returned by the Yahoo! search results. Since we do not have control over the search mechanism, these pipes may not be representative of the population. To reduce this threat, we obtained a large sample for analysis.

Along these same lines, the observations on individual authors only considered the most prolific authors and the pipes they submitted to the public repository that happen to be within our sample. This does not account for all the pipes created by these authors, nor all the pipes they submitted to the repository.

9.2. Construct

In our diversity analysis, we consider only structural similarity and distance, but not semantic similarity. Pipes with the same structure may not be similar at all. Future work is needed to evaluate if these are appropriate measures for capturing true diversity. While using two orthogonal diversity metrics controls this threat in part, it still remains. One future avenue to explore might be to refactor the pipes to remove some structural variability and recluster, or to measure the diversity of data sources or tags as an indication of semantics.

We use configurability as a proxy for abstraction, size for complexity, and clones to represent popularity or impact on the community. For clones, it was not possible to distinguish between self-cloning (which should not contribute to a pipe's popularity) and cloning by others. Future work is needed to control for this threat and evaluate if these are the most appropriate measures for the concepts we aimed to capture.

Our use of local and community uniqueness to draw conclusions about author awareness regarding pipe submissions introduces another threat. For high diversity and distance, we were unable to distinguish between coincidental and intentional uniqueness of submitted pipes.

9.3. Internal

The most clear internal threat is that all the analysis was done through the lens of a public repository, which offers limited visibility.

Some threats arise based on our selection criteria for the artifacts. For the author analysis, we selected prolific authors who had more than 15 pipes submitted within our sample. It is possible that this threshold is not the optimum to identify prolific authors. It may be these authors are not representative of the prolific authors and a different threshold should have been chosen. The clone counts were gathered at the time each pipe was scraped from the repository, and so the clone values among the pipes were often collected on different days, and the older pipes benefit from more time to have been cloned.

We do not measure how functional the submitted pipes are, nor can we tell the provenance of any pipe and so the complex structures may not have originated with the submitting author.

Another internal threat concerns the correctness of the tools we have developed, including the infrastructure to obtain and analyze the artifacts. While we have developed unit tests for all analyses and manually verified anomalies and interesting points in the data, the threat remains.

10. Conclusion

In this work, we present an artifact-based analysis of an end-user community, observing how authors evolve with time and the impact of different variables such as time and proliferation on the diversity, popularity, size, and configurability of artifacts in the Yahoo! Pipes public repository. Similar to other communities, there is high attrition and the contributions of the participants follow a skewed distribution where few of the authors are responsible for a majority of the artifacts. However, authors who stay active with the community seem to evolve. We have observed that more experienced authors tend to make pipes that are more diverse, unique, popular, large, and configurable than authors with less experience, and that only 10–30% of the pipes submitted by the most prolific authors are highly unique compared to their previous contributions and to the community. From the results of our analysis, we have identified several implications for how end users could be better supported as they interact with the Yahoo! Pipes language and community, with the hope that some of these findings can be extended to other end-user communities.

Acknowledgment

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

References

[1] K.T. Stolee, S. Elbaum, A. Sarma, End-user programmers and their communities: an artifact-based analysis, in: Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement, ESEM '11, 2011, pp. 147–156.
[2] C. Scaffidi, M. Shaw, B. Myers, Estimating the numbers of end users and end user programmers, in: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, VLHCC '05, 2005, pp. 207–214.
[3] R. Panko, What we know about spreadsheet errors, J. End User Comput. 10 (1998) 15–21.
[4] Yahoo! Pipes, February 2011. <http://pipes.yahoo.com/>.
[5] IBM Mashup Center, August 2009. <http://www.ibm.com/software/info/mashup-center/>.
[6] xFruits, August 2009. <http://www.xfruits.com/>.
[7] Apatar, August 2009. <http://www.apatar.com/>.
[8] CoScripter, February 2011. <http://coscripter.researchlabs.ibm.com/>.
[9] Userscripts, February 2011. <http://userscripts.org/>.
[10] Scratch, February 2011. <http://scratch.mit.edu/>.
[11] Kodu Game Lab, August 2010. <http://research.microsoft.com/en-us/projects/kodu/>.
[12] W. Scacchi, Free/open source software development: recent research results and methods, Adv. Comput. 69 (2007) 243–295.
[13] G. Von Krogh, S. Spaeth, K. Lakhani, Community, joining, and specialization in open source software innovation: a case study, Res. Policy 32 (7) (2003) 1217–1241.
[14] K. Crowston, K. Wei, J. Howison, A. Wiggins, Free/libre open source software development: what we know and what we do not know, ACM Comput. Surv. (2010) 7:1–7:35.
[15] N. Ducheneaut, Socialization in an open source software community: a socio-technical analysis, Comput. Sup. Cooperat. Work 14 (2005) 323–368.
[16] M.C. Jones, E.F. Churchill, Conversations in developer communities: a preliminary analysis of the yahoo! pipes community, in: Proceedings of the Fourth International Conference On Communities and Technologies, 2009, pp. 195–204.
[17] A.J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M.B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck, The State of the Art in End-User Software Engineering, ACM Computing Survey
[18] S.K. Kuttal, A. Sarma, A. Swearngin, G. Rothermel, Versioning for mashups: an exploratory study, in: Proceedings of the Third International Conference on End-User Development, IS-EUD'11, 2011, pp. 25–41.
[19] A.J. Ko, B.A. Myers, Debugging reinvented: asking and answering why and why not questions about program behavior, in: Proceedings of the 30th International Conference on Software Engineering, ICSE '08, 2008, pp. 301–310.
[20] A. Koesnandar, S. Elbaum, G. Rothermel, L. Hochstein, C. Scaffidi, K.T. Stolee, Using assertions to help end-user programmers create dependable web macros, in: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, SIGSOFT '08/FSE-16, 2008, pp. 124–134.
[21] K.T. Stolee, S. Elbaum, Refactoring pipe-like mashups for end-user programmers, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, 2011, pp. 81–90.
[22] C. Scaffidi, C. Bogart, M. Burnett, A. Cypher, B. Myers, M. Shaw, Predicting reuse of end-user web macro scripts, in: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), VLHCC '09, 2009, pp. 93–100.

[23] A. Dahotre, Y. Zhang, C. Scaffidi, A qualitative study of animation programming in the wild, in: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '10, 2010, pp. 29:1–29:10.
[24] JackBe, September 2012. <http://www.jackbe.com/>.
[25] L. Grammel, M.-A. Storey, The Smart Internet, 2010, pp. 137–151 (Chapter A survey of mashup development environments).
[26] F. Daniel, C. Rodriguez, S. Roy Chowdhury, H.R. Motahari Nezhad, F. Casati, Discovery and reuse of composition knowledge for assisted mashup development, in: Proceedings of the 21st International Conference Companion On World Wide Web, WWW '12 Companion, 2012, pp. 493–494.
[27] A.V. Riabov, E. Boillet, M.D. Feblowitz, Z. Liu, A. Ranganathan, Wishful search: interactive composition of data mashups, in: Proceeding of the 17th International Conference on World Wide Web, WWW '08, 2008, pp. 775–784.
[28] O. Greenshpan, T. Milo, N. Polyzotis, Autocompletion for mashups, Proc. VLDB Endow. 2 (1) (2009) 538–549.
[29] S. Soi, M. Baez, Domain-specific mashups: from all to all you need, in: Proceedings of the 10th International Conference on Current trends In Web Engineering, ICWE'10, 2010, pp. 384–395.
[30] I. Muhammad, D. Florian, C. Fabio, M. Maurizio, Reseval mash: a mashup tool that speaks the language of the user, in: Proceedings of the 2012 ACM Annual Conference Extended Abstracts on Human Factors in Computing Systems Extended Abstracts, CHI EA '12, 2012, pp. 1949–1954.
[31] A. Bozzon, M. Brambilla, M. Imran, F. Daniel, F. Casati, Search Computing, 2011, pp. 192-200 (Chapter On development practices for end users).
[32] R.J. Ennals, M.N. Garofalakis, Mashmaker: mashups for the masses, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of data, SIGMOD '07, 2007, pp. 1116–1118.
[33] G. Fischer, A. Piccinno, Y. Ye, The ecology of participants in co-evolving socio-technical environments, in: Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams, HCSE-TAMODIA '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 279–286.
[34] C. Bird, D. Pattison, R. D'Souza, V. Filkov, P. Devanbu, Latent social structure in open source projects, in: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, SIGSOFT '08/FSE-16, 2008, pp. 24–35.
[35] G. Madey, V. Freeh, R. Tynan, The open source software development phenomenon: an analysis based on social network theory, in: Proceedings of the Americas Conference on Information Systems (AMCIS 2002), 2002, pp. 1806–1813.
[36] W. Oh, S. Jeon, Membership herding and network stability in the open source community: the Ising perspective, Management Science 53 (7) (2007) 1086.
[37] J. Wong, J. Hong, What do we "mashup" when we make mashups?, in: Proceedings of the 4th International Workshop on End-User Software Engineering, WEUSE '08, 2008, pp. 35–39.
[38] JSON, August 2009. <http://www.json.org/>.
[39] M. Gabel, Z. Su, A study of the uniqueness of source code, in: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE '10, 2010, pp. 147–156.