Evolution of the mashup ecosystem by copying

Michael Weiss
Technology Innovation Management
SCE, Carleton University
Ottawa, Canada
weiss@sce.carleton.ca

Solange Sari
Technology Innovation Management
SCE, Carleton University
Ottawa, Canada
ssari@connect.carleton.ca

ABSTRACT

Previous work has explored the structure of the mashup ecosystem, which can be modeled as a network of mashups and APIs. However, it did not offer an explanation for its growth. In this paper, we seek an answer to the question how mashup developers select APIs. One hypothesis that has been put forward is that APIs are selected by their popularity, that is, by preferential attachment. However, this hypothesis is unsatisfying, as it disregards that mashups are composed from multiple APIs, and neglects the learning between mashup developers. Instead, we propose a copying model. We test to what degree developers create mashups by copying other mashups. We show that a good fit between the actual distribution of APIs and our model can be obtained. We can conclude that copying plays a significant role in explaining how mashups are developed. We also identify open research questions raised by the results.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services

General Terms

Experimentation, Measurement

Keywords

Web 2.0, Mashups, Web APIs, Copying, Evolution

1. INTRODUCTION

Mashups are applications that combine data and services provided by third parties through open APIs with user-supplied data [21]. For example, the Google Maps API generates maps for a given location, and its output can be combined with other open APIs and user-supplied data. Mashups allow the quick creation of custom applications short life span and specific context of use. This has given rise to a new application development model: opportunistic programming emphasizes speed and ease of development over robustness and ability to maintain the software [5].

The creation of mashups is supported by a complex ecosystem of interconnected data providers, mashup platforms, and users. Previous work has examined the structure of the mashup ecosystem and its growth over time [18, 20, 19]. Since late 2005, the evolution of the ecosystem of mashups and APIs has been documented public directories. One of these sources is the ProgrammableWeb site.[1] It lists APIs and mashups by date of introduction and profiles them. It also categorizes APIs and mashups through a provided taxonomy and tags that users can associate with the entries. Since the contents of the site are user-contributed, not all APIs and mashups in existence are indexed. However, the ProgrammableWeb is probably the most widely recognized mashup directory, and its contents can be considered representative of the state of the mashup ecosystem.

Previous work has explored the structure of the mashup ecosystem, but it did not offer an explanation for its growth. In this paper, we seek an answer to the question how mashup developers select APIs. One hypothesis that has been put forward is that APIs are selected by their popularity [20]. This is the classic preferential attachment hypothesis: the more links a node has, the more likely it is to be selected as target of a link by nodes added to the network [3, 4]. However, this hypothesis is unsatisfying for two reasons:

1. It seems to oversimplify the process by which mashups are created. While the popularity of an API can certainly be thought to influence the developer's choice, reducing the selection process to just that neglects other factors, such as category of mashup to be created (for example, search or travel), or how developers learn about APIs that can be combined.

2. It appears that mashup developers would learn from other developers. Just selecting the APIs that have been used most only captures part of this learning. If the lessons obtained from how people create web pages or how programmers learn a new programming language can provide any guidance [5], it is likely that developers will emulate each other.

We, therefore, propose to examine to what degree developers create mashups by copying other mashups. Like a new web page created by copying the HTML code from an existing web page [5], a new mashup would often start out as a copy

[1] http://www.programmableweb.com

of an existing mashup. However, with this analogy, there is an important difference. Whereas in the case of a web page, the source is usually available, a mashup author typically only has access to what APIs another mashup uses. Copying is restricted to the "blueprint" of the mashup, and does not include the logic of how the APIs are composed.

The rest of the paper is organized as follows. In Section 2, we review the related work on mashup ecosystems and copying as a design principle and a model of network growth. In Section 3, we introduce our copying hypothesis and describe a simulation model to test it. In Section 4, we outline our research method, and in Section 5 present the results of the simulation. In Section 6, we discuss the results and their implications. Section 7 concludes the paper.

2. RELATED WORK

In the following, we first review related work on the evolution of the mashup ecosystem, then the literature that makes a case for copying as a design principle, and finally existing work on copying as a model of network growth.

The structure of the mashup ecosystem and its growth over time has been examined in [18, 19] and [20]. The first study presents a research method for the analysis of mashup ecosystems [18, 19]. The authors develop techniques for visualizing the mashup ecosystem, and use network analysis to obtain characteristics of the ecosystem and to identify significant ecosystem members and their relationships. Similarly, the second study provides evidence that the mashup ecosystem is organized into three tiers, and that a small set of core APIs play a dominant role, but that peripheral APIs are key to the creation of new links in the network [20]. Yet, neither study offers insights into which internal mechanisms lead to the observed growth of the mashup ecosystem.

The ability to quickly assemble custom applications from existing parts has given rise to a new application development model. In opportunistic programming, speed and ease of development are given more weight than creating applications that are robust and maintainable [5]. The authors cite the example of web pages, which are often created by copying existing web pages, as an instance of opportunistic programming. Here, the concept of opportunism refers to the opportunities or options afforded by the components that can be quickly assembled into new applications. Similarly, open APIs are provided to users with the intent that users can create applications that had not been anticipated by API providers, and are often highly specialized to the needs of a few users. Open APIs enable users on the "long tail" of possible user needs to help themselves.

This perspective is consistent with research on the recombinant nature of the innovation process [8]. Innovation can be described as the construction of new ideas from existing ones. Benefits of recombination include shortening the learning curve by combining known elements in new ways, sharing of past experience across organizational boundaries, and the diversity of problem solving frames. The concept of recombinant innovation is closely linked to the concept of modularity. Modularity allows the creation of new products by mixing and matching components [6]. The increased modularity implied by open APIs is of great influence on the development of mashups. Open APIs are the modules that can be (re)combined into mashups. Modularity is also the basis for imitating the design of a mashup, when a user "clones" an existing mashup. Work on the role of imitation in innovation [7] leads us to conclude that modularity enables others to imitate the design of a system.

Many real network such as citation networks [15] and the Internet [2] have a degree distribution that observes a power law. Networks with a power law distribution are also known as scale free networks [2]. A growth model for scale-free networks has been proposed in [3] based based on two processes: growth (nodes are added continuously in the network) and preferential attachment (edges are added in proportion to the number of existing edges). Several mechanisms for generating power law distributions are described in [13].

The web growth model in [9, 10] describes a copying process that gives rise to a scale free network. The main step to their model is that new nodes are created by copying a subset of the links of a randomly selected existing node. Others [16, 17] also recognize that duplication mechanisms could explain the scale-free nature of biological networks. For example, the cell replication process has elaborate copying mechanisms to limit the number of replication errors. However, occasional errors are significant for creating the population diversity upon which selection acts to produce evolution.

3. COPYING MODEL

We model the mashup ecosystem as a network of mashups and APIs. Technically, this amounts to a bipartite graph $G = (M \cup A, E)$ with two sets of nodes, M, representing mashups, and A, representing APIs, E being the set of edges or links between the nodes. Mashup nodes $m \in M$ are only connected to API nodes $a \in A$. These are the APIs that are composed by the mashup author to provide the functionality of the mashup. For example, if a mashup m combines the Google Maps (a_{gm}) and the Flickr (a_f) APIs, the ecosystem graph will contain the edges (m, a_{gm}) and (m, a_f). The total number of nodes in the graph is $N = |M \cup A|$.

Our approach is to create a simulation model of the evolution of the mashup ecosystem under a copying hypothesis, and to compare the characteristics of the resulting network to those of the actual network. The initial model was created by starting with an algorithm for simulating the growth of the web proposed in [9], which was then iteratively adapted. We also implemented models for random growth and evolution by preferential attachment to compare with the model.

In our initial model, we made the assumption that each mashup has the same number (m) of APIs. In the future, we plan to modify the simulation model to let the number of APIs per mashup vary according to a distribution. The ratio $r = |A|/|M|$ of APIs to mashups was obtained from the data on the ProgrammableWeb.[2] The proportion p of APIs among network nodes is $p = |A|/|M \cup A|$.

Initialize the network. The initial network consists of m_0 nodes (where $m_0 \geq m$). The first $m_0 - 1$ nodes are created

[2]http://api.programmableweb.com

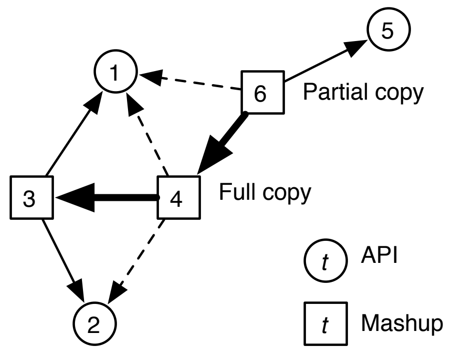


Figure 1: Illustration of how copying works

as APIs. The last node is a mashup that combines m of the APIs. This step is required as the network does not yet have m nodes from which the first mashup can be composed. From the related work, we can expect that the exact value of m_0 has little impact on the results. Previous work [9] showed that the size of the initial network does not affect the results in a significant way, as long as $N \gg m_0$.

Grow the network. APIs and mashups are added continually to the network. For simplicity, we assume that changes to the network occur in N discrete timesteps.[3] At each timestep t, starting at $t_0 = m_0 + 1$, a new API is added to the network with probability p. With probablity $1 - p$, an existing mashups m_s is selected from the set of mashups M_{t-1} at timestep $t - 1$. This mashup provides the template for a new mashup to be added to the network.

For each API in the template, the API is either copied to the new mashup or substituted by a random API. With probability α, the API is copied from the template. With probability $1 - \alpha$, a new API is chosen at random from the set of APIs A_{t-1} at timestep $t - 1$. This copying factor (α) is the proportion of copied APIs in a mashup. An α close to 1 implies that most APIs are copied from the template, whereas an α close to 0 means that most APIs are chosen at random. When APIs are chosen randomly, care is taken not to choose the same API twice per mashup.

Figure 1 illustrates how the copying model works. In this example, two APIs (nodes 1 and 2) and one mashup (node 3) that uses those APIs are created during the initialization phase.[4] The solid lines indicates that these links were randomly selected. Next, assume that mashup (node 4) is created as a copy of node 3. A thick solid line indicates a "copies" relationship. This mashups is a full copy of the mashup in node 3, so we add links to nodes 1 and 2. A dashed line indicates that these links were copied. The next node added is an API (node 5). Finally, we add a mashup (node 6) as a partial copy of node 4. The copy retains the link to node 1, but adds a link to node 5 at random.

4. RESEARCH METHOD

4.1 Calibration

The ProgrammableWeb maintains a list of all APIs in the directory sorted by their popularity. A snapshot of this list taken on August 16, 2010 shows that there were 2084 APIs. Only 703 of these have been used in a mashup at least once. A count of 0, however, does not mean that this API is not important, just that there are no records (yet) of mashups using this API in the directory. Thus, we include them in the sample. The 2084 APIs were used by 5028 mashups. The ratio r of APIs to mashups is, therefore, 0.414.

4.2 Simulation

The simulation model described in Section 3 was implemented in Perl. The analysis of the simulation results was conducted using the iGraph package for R.[5] iGraph offers functions for network analysis, including the computation of social network metrics (e.g. degree distribution), network visualization and fitting power law distributions. We added functions for computing the sum of squared error fit and for obtaining various characteristics of the mashup graph.

Network structure. For a given combination of input parameters ($m = 2$, $\alpha = 0.750$), Figure 2 shows snapshots of the simulated network after 100, 500, and 2500 timesteps. API are shown as circles, mashups as squares. The size of each node is proportional to the number of links it has. As the number of links follows a Zipf distribution, some nodes will have a disproportionally high number of links [1]. Thus, we base node size on the logarithm of its degree. If k_i is the degree of node i, we compute its size as $log(1 + k_i) + 1$.

Each simulation was run for $N = 7112$ timesteps, the combined number of APIs and mashups on the day for which we took the snapshot of the directory. The results of the simulation are affected by two parameters: the number of APIs in a given mashup m (mashup size), and the copying factor α, which determines to what degree the evolution of the mashup ecosystem is the result of copying.

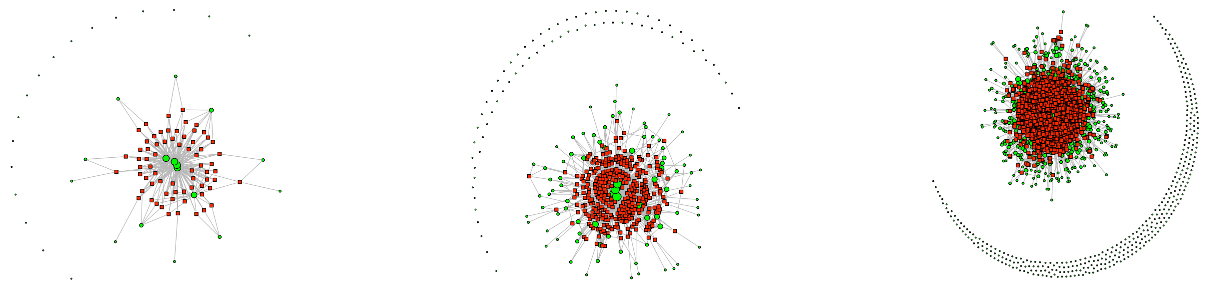
Each snapshot consists of a large component comprised of mashups and APIs, and a group of unused APIs, which are shown as a crescent around the center. However, not all mashups and APIs are necessarily in the central component. They can also form smaller components on their own. Different values of m and α will result in similar types of diagrams. They differ primarily in terms of their API degree distribution and the number of unused APIs.

Best fit with actual data. We systematically varied the input parameters for the simulation to arrive at a best fit between the distribution of the number of mashups per API that can be observed in the actual data with the simulated data. The 2084 APIs are used a total of 9833 times, or on average $9833/5028 = 1.96$ times per mashup. Therefore, a value of $m = 2$ appears to be a reasonable choice. A larger m would result in a higher than actual use of the APIs.[6]

Determining the best fit then becomes a task of finding the value of α that minimizes the difference between the actual

[3]Where $N = |M \cup A|$ is the number of nodes, as above.
[4]Node numbers correspond to timesteps.

[5]igraph.sourceforge.net
[6]This obviously simplifies our analysis.



(a) $N = 100$ (b) $N = 500$ (c) $N = 2500$

Figure 2: Snapshots of the simulated network after 100, 500, and 2500 timesteps ($m = 2$, $\alpha = 0.750$)

and simulated distributions. We explored two different ways to accomplish this: sum of squared error fit, and power law fit. In the first approach, our goal is to minimize the sum of squared error (SSE) between the degrees (that is, the number of mashups using an API) in the simulated and the actual mashup ecosystem networks:

$$SSE = \sum_{i \in A_{act} \cap A_{sim}} (k_i^{sim} - k_i^{act})^2 \tag{1}$$

A_{act} and A_{act} are the sets of APIs in the actual and simulated network, respectively. As the number of APIs in the actual and simulated network may differ, we only run the index over APIs that are included in both.[7] k_i is the degree of the ith API in the network. We assume the lists to be ordered by the rank of the API, starting with the most popular API. The superscript indicates whether we refer to a node in the actual or the simulated network.

In the second approach, we chose to determine the best fit as the value of α for which the closest correspondence between the Zipf exponent of the simulated and actual distributions is obtained. The distribution of API degrees follows a Zipf distribution [1]. A Zipf distribution is typically shown as a plot of the frequency of an event relative its rank. Zipf distributions have been used to describe the frequency of words in English text, the size of cities, or the number of visitors to web pages. Figure 3 shows the actual distribution of APIs measured in the mashup ecosystem. The line has a slope of -0.990, which is very close to the expected -1 slope linking frequency to rank in a typical Zipf distribution.

In a log-log plot of degrees over their rank, a Zipf distribution will be a straight line. Its slope can be estimated by fitting the distribution to a power law. A distribution is said to follow a power law, if it adheres to the form $P(x) \sim x^{-a}$. From this, we obtain the exponent b of the ranked plot of the Zipf distribution $y \sim r^{-b}$ by subtracting 1 from the coefficient a in the power law fit of the Zipf distribution. In a Zipf distribution, the coefficient a is approximately 2; hence, the exponent b will be around 1. To estimate the power law coefficient, we use the maximum likelihood technique as recommended in [11] and implemented in iGraph.

[7]Due to the long tail nature of the degree distribution, the nodes excluded from the calculation can be expected to have a degree of 1, leading to a negligible error term compared to the squared errors of nodes with higher degree.

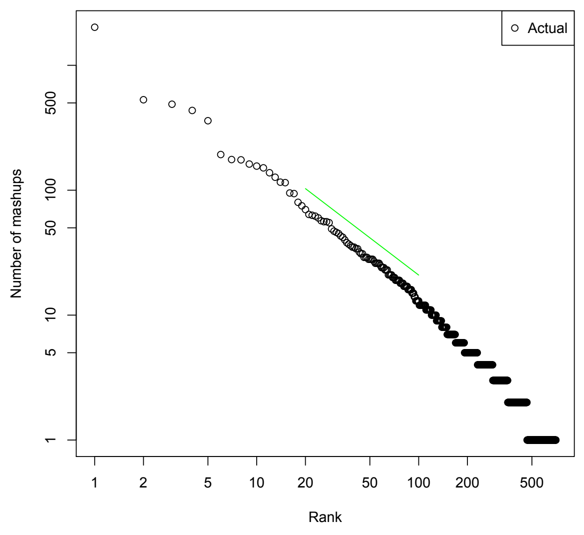
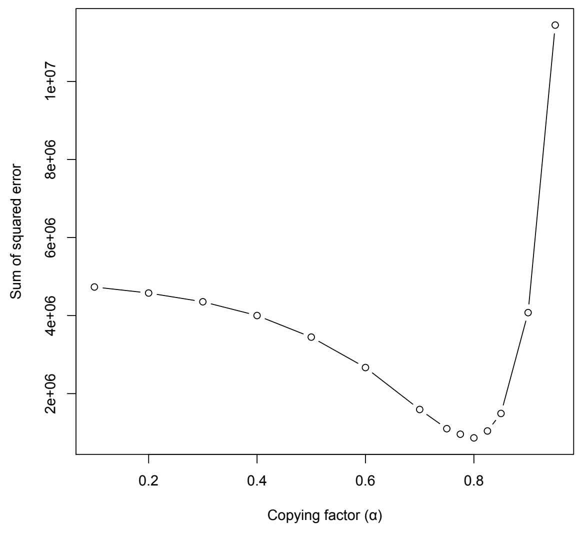
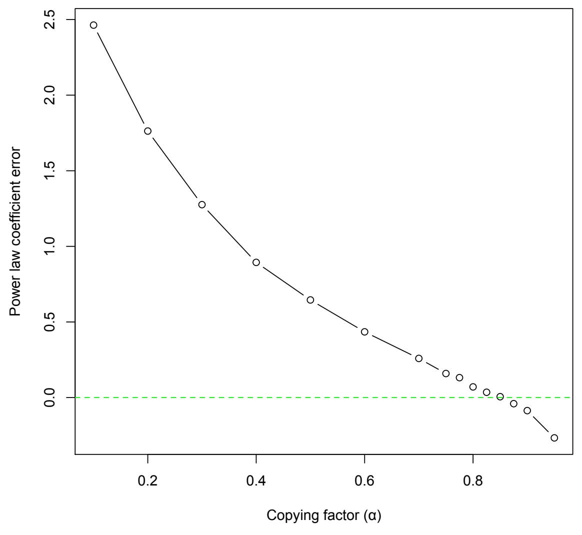


Figure 3: Actual distribution of the number of mashups for the APIs in the mashup ecosystem

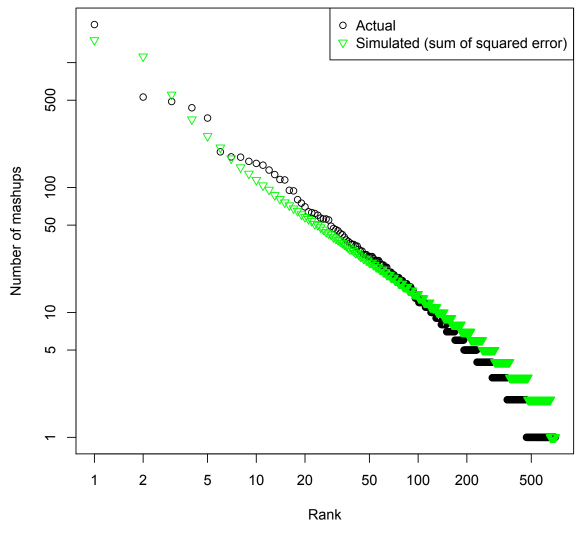
To increase the accuracy of the simulation results, we ran each simulation multiple times. To determine the appropriate number of simulation runs, we calculated the 95% confidence interval and used it to assess whether the accuracy required to carry out the minimization was sufficient. Initially, we conducted 30 simulations for each value of α, and increased it first to 100, then to 1000, respectively, as the difference between the value of α and its previous value decreased with the progress of the minimization.



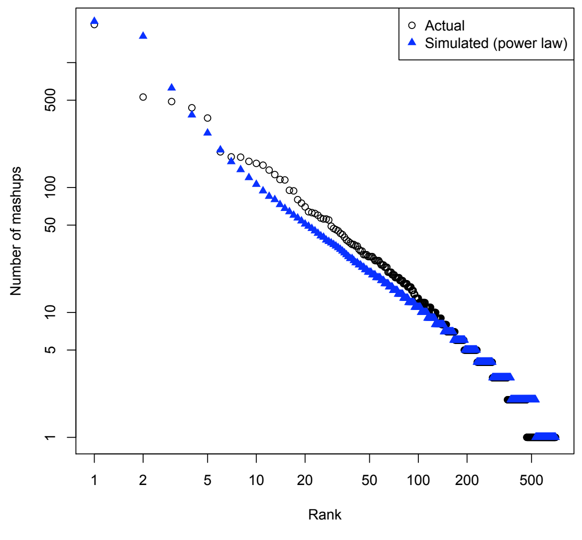
(a) Sum of squared error as a function of α



(b) Power law fit error as a function of α



(c) Sum of squared error fit for $m = 2$, $\alpha = 0.798$



(d) Power law fit for $m = 2$, $\alpha = 0.855$

Figure 4: Simulation results for the copying model

5. RESULTS

5.1 Sum of squared error fit

Figure 4(a) shows the sum of squared error for different values of α. The best fit occurs for $\alpha = 0.798$. The power law coefficient for the simulated network is 2.089. Thus, the Zipf plot has a slope of -1.089. Figure 4(c) shows the Zipf plot for the sum of squared error fit. We observe that the simulation underestimates the number of mashups for the top-ranked API (Google Maps) and overestimates the number of mashups for the second-ranked API (Flickr).

Two other indicators are the number of APIs used by at least one mashup, and the number of APIs that contributed to 50% of API uses in mashups. In the sum of squared error fit, there are an average of 1020 APIs with more than one mashup as compared to 703 in the actual distribution.

In the actual distribution, the 12 top APIs contribute 50% of the APIs uses in mashups (4981 out of 9833). By comparison, in the sum of squared error fit, it takes 16 APIs to achieve 50% (5079 out of 10156). Thus, the sum of squared error fit underestimates the contributions of the top APIs. Figure 5 shows the cumulative contributions of APIs in the simulated network for the sum of squared error fit.

5.2 Power law fit

Figure 4(b) shows the power law fit error. The best fit occurs for $\alpha = 0.855$. While the value is different from that obtained for the sum of squared error fit, the values are close together. The power law coefficient for this simulated network is 1.986 with a 95% confidence interval of $(1.983, 1.990)$. The ranked Zipf plot shown in Figure 4(d) has a slope of -0.986. We can see that, unlike the sum of squared error fit, a power law fit error overestimates the number of mashups for the top-ranked API, but similarly overestimates the number of mashups for the second-ranked API.

Again, it is interesting to look at the number of APIs with more than one mashup and the cumulative contribution of APIs. In the power law error fit, there are an average of 859 APIs with more than one mashup as compared to 703 in the actual distribution. In the power law error fit, it takes 5 APIs to achieve 50% (5044 out of 10152). Thus, the power law error fit overestimates the contributions of the top APIs. Figure 5 shows the cumulative contributions of APIs in the simulated network for the power law error fit.

6. DISCUSSION

Both the sum of square error and power law error fits approaches obtained their best fit for a high copying factor. This suggests that most mashups are created by using an existing mashup as a template and modifying its design. In 79.8% to 85.5% of the cases, an API in the template mashup is copied to the new mashup, otherwise it is substituted by a different API. *This indicates that copying plays a significant role in the evolution of the mashup ecosystem.*

As expected, a power law fit error more closely approximates the actual power law distribution than a sum of squared error fit. However, the sum of square error fit provides a closer match of the actual degrees of the APIs in the midrange of the distribution (ranging approximately from rank 20 to

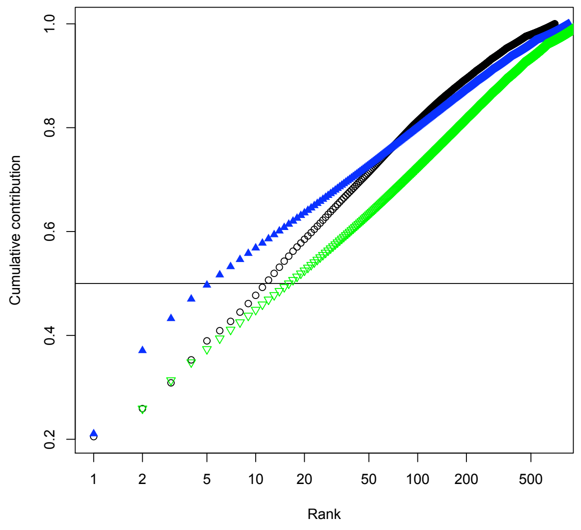


Figure 5: Cumulative contribution of APIs to mashups

100). The former overestimates the number of mashups for the top-ranked API, the latter underestimates it.

To further examine this effect, we studied the contributions of APIs to mashups. While the sum of squared error fit underestimates the contributions of the top APIs (that contribute 50% of the uses), the power law error fit overestimates it. However, neither provided a close estimate of the actual value. In part, this can be attributed to the overestimation of the contribution of the second-ranked API.

7. CONCLUSION

The results of our simulation suggest that copying plays a significant role in the evolution of the mashup ecosystem. However, we cannot rule out that other factors are at play that could explain how the mashup ecosystem grows.

Future research should test the copying hypothesis empirically. One suggested way to go forward is to examine the relationships between mashups in the Programmable Web data. Our current work on evolutionary processes in the mashup ecosystem heads that way. Another approach would be to examine online repositories of mashups such as Yahoo Pipes that support copying.[8] The Yahoo Pipes platform provides a repository where developers can share the mashups they created. A new pipe can be built by explicitly cloning an existing mashup. Thus, a link between the existing mashup and the clone is established. This offers an interesting venue for studying copying processes *in situ*.

Another open question is the link between copying and diversity in the mashup ecosystem. We might expect copying to lead to a more homogeneous landscape of mashups. If new mashups imitate the design of existing mashups would this not reduce the diversity of solutions and, thus, the amount

[8]http://pipes.yahoo.com

of innovation in the mashup ecosystem? Our current work tries to address this question by applying models from evolutionary biology. This research shows that the evolution of species can be represented as a phylogenetic tree [14]. Such a tree shows the evolutionary relationships among species that can be reconstructed from similarities and differences in their characteristics [12]. Birth-death models allow us to estimate the diversity rate from a phylogenetic tree.

Acknowledgements

Thanks to John Musser for providing an API key to access the data on the Programmable Web. Thanks for the students in the Technology Innovation Management program for their stimulating discussions related to this research.

8. REFERENCES

[1] L. Adamic, and B. Huberman. Zipf, Power-laws, and Pareto - a ranking tutorial. HP Labs, 2000.

[2] R. Albert, H. Jeong, and A.L. Barabasi. Diameter of the World Wide Web. *Nature*, 401, 130-130, 1999.

[3] A.L. Barabasi, R. Albert. Emergence of scaling in random networks. *Science*, 286, 509-512, 1999.

[4] A.L. Barabasi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A*, 311(3-4), 590-614, 2002.

[5] J. Brandt, P. Guo, J. Lewenstein, S. Kelmmer, and M. Dontcheva. Opportunistic programming: Writing code to prototype, create, and discover. *IEEE Software*, September/October, 18-24, 2009.

[6] S. Ethiraj, and D. Levinthal. Modularity and innovation in complex systems. *Management Science*, 50(2), 159-173, 2004.

[7] S. Ethiraj, D. Levinthal, and R. Roy. The dual role of modularity: Innovation and imitation. *Management Science*, 54(4), 939-955, 2008.

[8] A. Hargadon. Brokering knowledge: Linking learning and innovation. *Research in Organizational Behavior*, 24, 41-85, 2002.

[9] J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. 1999. The Web as a Graph: Measurements, Models, and Methods. *International Conference on Computing and Combinatorics*, LNCS 1627, Springer, 1-17, 1999.

[10] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins and E. Upfal. The web as a graph. *ACM Symposium on Principles of Database Systems*, 1-10, 2000.

[11] M. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46, 323-351, 2005.

[12] S. Nee, R.M. May, and P.H. Harvey. The reconstructed evolutionary process. *Philosophical Transactions of the Royal Society of London*, Series B, Biological Science, 344, 305-311, 1994.

[13] M. Newman. *Networks: An Introduction.* Oxford University Press, 2010.

[14] E. Paradis. *Analysis of Phylogenetics and Evolution with R.* Springer, 2006.

[15] D. J. de S. Price. Networks of scientific papers. *Science*, 149, 510-515, 1965.

[16] R.V. Sole, R. Pastor-Satorras, E. Smith, and T.B. Kepler. A model of large-scale proteome evolution. *Advances in Complex Systems*, 5(1), 43-54, 2002.

[17] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Global protein function prediction from protein-protein interaction networks. *Nature Biotechnology*, 21, 697-700, 2003.

[18] M. Weiss, and G.R. Gangadharan. Innovation in mashup ecosystems. *R&D Management Conference*, 2008.

[19] M. Weiss, and G.R. Gangadharan. Modeling the mashup ecosystem: Structure and growth. *R&D Management*, 40(1), 40-49, 2010.

[20] S. Yu, and J. Woodard. Innovation in the Programmable Web: Characterizing the mashup ecosystem. *Workshop on Web APIs and Services Mashups*, LNCS 5472, Springer, 136-147, 2008.

[21] J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *IEEE Internet Computing*, September/October, 44-52, 2008.