Social Coverage for Customized Test Adequacy and Selection Criteria

Breno Miranda
Università di Pisa
Largo B. Pontecorvo, 3 - 56127
Pisa, Italy
miranda@di.unipi.it

Antonia Bertolino
ISTI - CNR
Via Moruzzi 1 - 56124
Pisa, Italy
antonia.bertolino@isti.cnr.it

ABSTRACT

Test coverage information can be very useful for guiding testers in enhancing their test suites to exercise possible uncovered entities and in deciding when to stop testing. However, for complex applications that are reused in different contexts and for emerging paradigms (e.g., component-based development, service-oriented architecture, and cloud computing), traditional coverage metrics may no longer provide meaningful information to help testers on these tasks. Various proposals are advocating to leverage information that come from the testing community in a collaborative testing approach. In this work we introduce a coverage metric, the Social Coverage, that customizes coverage information in a given context based on coverage data collected from similar users. To evaluate the potential of our proposed approach, we instantiated the social coverage metric in the context of a real world service oriented application. In this exploratory study, we were able to predict the entities that would be of interest for a given user with an average precision of 97% and average recall of 75%. Our results suggest that, in similar environments, social coverage can provide a better support to testers than traditional coverage.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools (e.g., data generators, coverage testing)*

General Terms

Measurement, Verification

Keywords

Coverage Testing, Relative Coverage, Service-Oriented Application, User Similarity

1. INTRODUCTION

Test coverage information can be very useful for guiding testers in enhancing their test suites so to exercise possibly uncovered parts of the program and in deciding when to stopping testing. Although there is no consensus about the relationship between code coverage of a given test suite and its ability to reveal faults, having coverage information is still important as it is evident that a test suite can hardly find bugs in code that is never executed.

The adequacy criteria proposed in the literature are typically based on the same underlying principle: a set of entities that must be covered (they could be statements, branches, paths, operations, and so on) is identified, and a program is not considered to be adequately tested if some entities have never been executed by any test data. Coverage is measured as the ratio between the covered entities and the total number of entities in the program under test. However, the application of this traditional way of measuring coverage to complex systems that are reused in different contexts might not always provide meaningful information. In fact, not all entities might be of interest in every context.

This is particularly true for several new programming paradigms emerged in the last decade: component-based development, service-oriented architecture (SOA), and cloud computing are just a few examples. In many cases, the reasons that make these paradigms attractive also make them more challenging to test. In the SOA domain for example, the separation between interface and code is a key feature behind the highly appreciated flexibility of this technology. Several researchers however have remarked as such separation also leads to low testability [6, 5].

Speaking of coverage testing, a same service makes available many operations that may be invoked by different testers for different purposes and in different combinations. In operation coverage testing, an operation is covered if, during the service execution, it is called at least once. Traditional coverage would indifferently assess the operations invoked by each of those testers against the full set of operations made available, which may provide a misleading low ratio.

This work is inspired by the idea of *Relative Coverage*, originally defined in [3], and subsequently exploited in [8]. Relative coverage introduces the notion of a flexible and context-dependent test adequacy measure. Differently from traditional coverage, relative coverage adapts to a tester's context, and measures the ratio between the covered entities and those that are considered relevant in the given situation and environment, which do not necessarily comprise the full set of available entities.

This is an attractive idea, but brings the difficult challenge on how to identify what are the relevant entities from time to time (i.e., what is the denominator in the coverage ratio).

AST'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2858-6/14/05...$15.00
http://dx.doi.org/10.1145/2593501.2593505

The cited works [3, 8] solved this challenge by requiring the testers themselves to specify explicitly the entities of interest. However this information may not be easily a-priori known by testers.

In this work we propose a method for identifying the relevant entities that is inspired by the recent trend of collaborative testing of web services [2], [24], [28]. Our coverage metric, called the *Social Coverage*, customizes coverage information in a given context leveraging information from coverage data collected from similar users. In our exploratory study on a real-world service-oriented infrastructure we obtain some preliminary evidence that information on system usage from service users that behave similarly to the tester can provide more meaningful test guide than traditional coverage approaches. Our proposed approach can mitigate the limitations of traditional coverage and can provide meaningful coverage results to testers, guiding them towards increasing test coverage in the areas that are relevant to them.

In summary, the contribution of this paper includes the definition of a novel notion of relative coverage, called social coverage; an approach to measure social coverage that leverages coverage information obtained from similar users to automatically identify the targeted entities; and an exploratory study on a real-world system assessing social coverage accuracy in predicting a tester's targeted entities.

In the remainder of the paper, Section 2 motivates our approach through an example scenario; Section 3 defines and illustrates the approach application on the motivating scenario; Section 4 reports the results from an exploratory study on the data log of the gCube real-world system; finally, Sections 5 and 6 discuss related work, conclusions and future work.

2. MOTIVATING SCENARIO

To introduce the notion of Social Coverage, we will use the example of a service provider S implementing a Travel Reservation System (TRS). As illustrated in Figure 1, TRS includes several features, relative to flight and hotel booking, car reservation, user registration, login and payment. These features can be accessed via a set of operations (in total 42 for our illustrative example) that are made available through the service public interface. Many operations can be associated with a same feature (e.g., the flight booking feature can contain the operations `FlightLookup`, `CheckSeatAvailability`, `getFlightTicketPrice`, and so on).

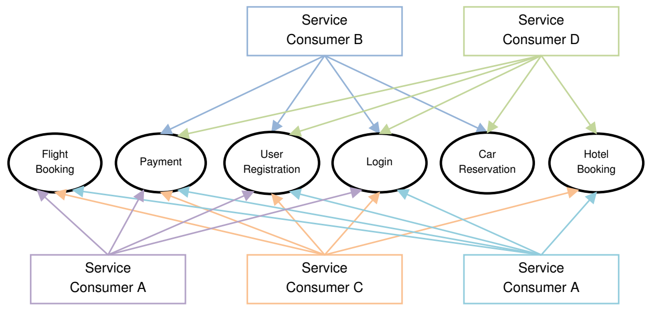

Figure 1: Travel Reservation System implemented by service provider S.

TRS services can be used by many consumers. In the example in Figure 1 we show five different service consumers, each using different combinations of the provided operations. Service consumer A, for example, uses operations from the flight booking, payment, user registration, and login features, suggesting that it is a service being used in a travel agency. Service consumer B, on the other hand, adopts operations from car reservation, user registration, login, and payment, suggesting that the service is being used by a car rental company. Service consumers A and B are different clients making use of the same service provider.

Let us now assume that a new service, say service consumer N, is developed. It is used in a travel agency and, as depicted in Figure 2, it is implemented to invoke the operations available from: flight booking, hotel booking, user registration, login, and payment. To diligently test the integration of the new service N with TRS, a tester creates a test suite to exercise the operations implemented by the new service N as well as the operations used from the service provider S.

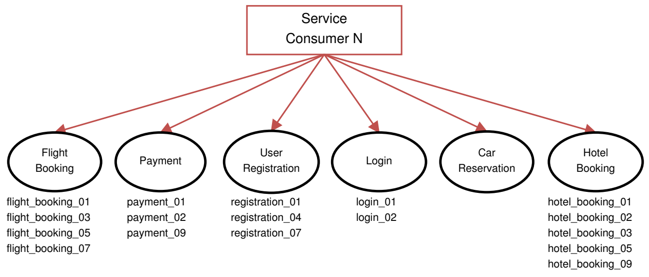

Figure 2: Service consumer N.

Let us assume that the test suite covers a total of 17 operations (those listed in Figure 2), and the tester is interested in assessing coverage over S (in terms of operation coverage). Traditional coverage, as seen in Equation 1, would be calculated by dividing the amount of operations invoked by N by the total amount of operations available in the service provider S.

$$\text{Traditional coverage} = \frac{\#\ covered\ entities}{\#\ available\ entities} \Rightarrow \frac{17}{42} \approx 40\% \tag{1}$$

Such a coverage metric would be helpful if the tester were interested in thoroughly testing the service provider S considering *all* the operations, including the ones that are never used by N, such as those in the car reservation feature. If, on the other hand, the tester is deliberately not interested in some operations, which, in fact, is the case in this example, this coverage information would not be meaningful.

Even worse, let us suppose that, besides calculating the ratio of covered operations, service N would also like to get the list of un-tested operations among those provided by S. Again, this list would be of little value if it cannot help testers to identify possibly relevant operations that have been left un-tested. In this very simple example, using the traditional approach to measure coverage, such list would contain among un-tested operations also those from car reservation; for really large applications, the number of not relevant operations could be so big that providing such information would be useless.

For the testing of N, we would like to measure coverage over the actually relevant operations, and not over the whole set of 42 operations. We introduce the intuitive notion of *targeted entities*, as those entities that are of relevance to a tester when measuring coverage on a provided service S.

The approach we present in the next section assumes that in an effort to better support customers in their testing activities, service provider S is willing to provide relevant coverage information to them. One way of doing that would be customizing coverage information for customer tester based on what operations have been invoked by service consumers that are "similar" to the service consumer under test. Thus, we call it the *Social Coverage* testing approach.

3. SOCIAL COVERAGE

To overcome the limitations of traditional coverage illustrated in Section 2, we propose the social coverage measure that customizes coverage information in a given context based on coverage data collected from similar users.

3.1 Approach

As for any coverage criterion, social coverage measurement presupposes that the code is instrumented so to allow the identification of the entities exercised by customers (they could be humans or other software).

For a target tester (i.e., the tester who is interested in receiving coverage information), social coverage measurement can then be summarized in the following steps:

1. **Data collection**: the entities exercised by the target tester are monitored. In addition, for social coverage testing it is assumed that usage information, including the list of entities exercised, are being routinely collected for all customers using the system under test.

2. **Similarity calculation**: the similarity between the target tester and all other system customers is calculated.

3. **Identification of *targeted entities***: the list of targeted entities is obtained by combining information about the entities exercised so far by the tester and the ones exercised by *similar* system customers.

4. **Coverage measurement**: social coverage is calculated and the results are provided to the target tester.

5. **Coverage increase**: a list of possibly relevant entities that have not been exercised by the target tester is suggested.

To better explain the approach, in the next section we apply it to the motivating scenario depicted in Section 2.

3.2 Illustration

Owning the instrumented code, the service provider S can get detailed information about the operations used by each of its service consumers. As the first step of our approach, using this information the service provider S can calculate the similarity between the service N and all the other service consumers to find the most similar services.

Many similarity measures could be adopted to find similar service consumers: Jaccard similarity coefficient [16], Pearson Correlation [13], and Euclidean Distance [22] are just a few examples. At this stage, we do not go deep in the details of each technique as we are still investigating which one would be the most adequate to our purpose.

For this illustrative example and for the exploratory study presented next, we used the Jaccard similarity coefficient. This decision was taken based on the results achieved by Hemmati and Briand [16]. In this work, the authors compared the cost of different similarity functions both in terms of computational complexity and the actual time required for the similarity calculation, when applied to the task of test case selection, and Jaccard was shown to be the most cost-effective similarity measure.

Table 1 reports the Jaccard similarity coefficient (presented as percentage) between service N and the other service consumers. The table shows that, based on the operations used by each client, service consumer E is the most similar service when compared to service N with a similarity degree of 88%; service consumer C is the second most similar service with a similarity degree of 70%; and so on.

Having calculated the similarities, the next step is identifying what are the targeted entities for service consumer N, that is, the operations that might be of interest to service consumer N and should be considered when calculating coverage (the denominator of Equation 2). For this task, it is important to define a similarity threshold to guarantee that only the most similar service consumers are considered when defining the set of targeted entities.

To continue with our example, let us assume that we are interested in analysing only the service consumers whose similarity with N is at least 70%. In this case, services A, B, and D would be discarded. As we are using the Jaccard similarity coefficient in this example, a simple way to define the entities targeted by N would be considering the union (or the intersection) of the operations invoked by the most similar service consumers. The adoption of collaborative filtering technique [21], on the other hand, would be more powerful as it would allow us not only to identify the set of entities exercised by similar clients, but also to estimate the probability that the service N is interested in a given entity. Such improvements will be deeper investigated in future work.

Using the union of the operations exercised by similar service consumers, the set of targeted entities for N is composed by 23 operations: all operations from service consumer C plus `hotel_booking_09` (invoked by E). We can now calculate the social coverage according to Equation 2:

$$\text{Social coverage} = \frac{\#\ \textit{covered entities}}{\#\ \textit{targeted entities}} \Rightarrow \frac{17}{23} \approx 74\% \qquad (2)$$

In comparison with traditional coverage calculated according to Equation 1, this is a more meaningful measure: the point is not in merely achieving a higher score, but in having a more realistic estimate of what could be achieved by augmenting the test suite.

Moreover, service S can provide a list of possibly relevant operations that have not been invoked by N (see Table 2). Since the list shown in Table 2 has been customized for N according to the behavior of similar services, it can be much more helpful in guiding testers to decide whether or not they need to increase the coverage of their test suites than simply providing the full list of operations available in the system. In the case the tester decides to create new test cases to cover previously un-tested operations, steps 1 to 5 of our

Table 1: Similarity calculation for service consumer N.

Service Consumer	**Service A**	**Service B**	**Service C**	**Service D**	**Service E**
Covered Entities	flight_booking_01	payment_01	flight_booking_01	payment_01	flight_booking_01
	flight_booking_02	payment_02	flight_booking_03	payment_02	flight_booking_03
	flight_booking_04	payment_03	flight_booking_05	payment_09	flight_booking_07
	flight_booking_07	payment_09	flight_booking_07	registration_01	payment_01
	flight_booking_08	registration_01	payment_01	registration_04	payment_02
	payment_01	registration_04	payment_02	registration_07	payment_09
	payment_02	registration_05	payment_03	login_01	registration_01
	payment_09	registration_07	payment_04	login_02	registration_04
	registration_01	login_01	payment_05	car_reservation_03	registration_07
	registration_04	login_02	payment_09	car_reservation_04	login_01
	registration_07	car_reservation_02	registration_01	car_reservation_07	login_02
	login_01	car_reservation_03	registration_02	hotel_booking_01	hotel_booking_01
	login_02	car_reservation_05	registration_03	hotel_booking_02	hotel_booking_03
		car_reservation_07	registration_04	hotel_booking_05	hotel_booking_05
			registration_05	hotel_booking_08	hotel_booking_09
			registration_07		
			login_01		
			login_02		
			hotel_booking_01		
			hotel_booking_02		
			hotel_booking_03		
			hotel_booking_05		
Similarity with N	50%	35%	70%	52%	88%

Table 2: Possibly relevant operations that have not been invoked by N.

Un-tested Operations
payment_03
payment_04
payment_05
registration_02
registration_03
registration_05

approach are repeated and social coverage information is updated according to the new data available.

4. EXPLORATORY STUDY

In this section we share our experience in instantiating the social coverage metric in the context of a real world system. The Research Question addressed in this exploratory study is to what extent would our approach be able to accurately predict the operations that would be relevant to a given user based on the analysis of the operations invoked by similar users. Details are given in the next sections.

4.1 Study Setup

Our study was carried out in the context of gCube, a Service Oriented Infrastructure enabling software, supporting the definition of Virtual Research Environments (VREs). A VRE is a flexible and agile application development model based on the notion of Software as a Service (SaaS), in which components may be bound instantly, just for the time they are needed, and then the binding may be discarded. gCube has been implemented with the support of the European Commission in the context of a series of projects [12].

gCube was apt for experimenting our social coverage approach, since it is equipped with a monitoring tool based on a messaging system that logs, among other things, the usage of some of the available services accessed through a web portal. In particular, the log from the D4Science infrastructure [1], enabled by gCube, was used to carry out our study.

Whenever a portal operation is called, the following information are stored: the operation called, the user that made the call, the date and time, the calling scope, and the message exchanged on that call. These data are made available through a consumer library. Such library allows the user to retrieve database content, exposes facilities to parse results as JSON objects, and aggregate information for the creation of statistics.

The steps below summarize the procedure followed in this study:

1. Collect usage data from D4Science portal using the consumer library API

2. Cleanup the data collected by removing *fake* users (used for testing purposes), portal administrators, and users with very few operations invoked

3. For each remaining user:

   (a) Split the data generated by that user into *training* and *testing* sets

   (b) Use the data available in the training set to predict what entities would be relevant to that user

   (c) Use the information from the testing set to compute the *precision* and *recall* of the predictions made

Table 3: Precision and Recall (training set = 10 invocations).

User	# Similar Users	Predicted Entities	List of Actually Covered Entities	Precision	Recall
...	...	...	...	...	...
U_025	5	ga_i_loginrecord, hlrecord, loginrecord	ga_i_hlrecord, ga_i_loginrecord, hlrecord, loginrecord	100%	75%
U_026	2	loginrecord, ebob_sl_loginrecord, hlrecord, ebob_sl_hlrecord, loginrecord	loginrecord, ebob_sl_loginrecord, hlrecord, ebob_sl_hlrecord	80%	100%
U_027	10	hlrecord, loginrecord	ga_t_loginrecord, hlrecord loginrecord	100%	66%
U_028	11	ds_dvre_loginrecord, ebob_loginrecord, ebob_sl_hlrecord, ebob_sl_loginrecord, hlrecord, loginrecord	ebob_loginrecord, ebob_sl_hlrecord, ebob_sl_loginrecord, hlrecord, loginrecord	83%	100%
U_029	51	hlrecord, loginrecord	hlrecord, loginrecord	100%	100%
...	...	...	...	...	...
			Overall Average:	**99%**	**72%**

4.2 Preliminary Results

We collected usage data (step 1) from 2009-11-05 to 2013-11-28, which resulted in 94.768 invocations logged from a total of 484 portal users. In the cleaning up phase (step 2), we noticed that many users had very little information logged. For example, about half of the users had only one invocation logged. When working with predictions/recommendations, such situations could be affected by the cold-start problem [18], the very well-known issue that a given system cannot draw any inferences for users about whom it does not have sufficient information.

Defining the minimum amount of information required for obtaining good predictions/recommendations is a hard task. This is an acknowledged problem in the recommender systems literature [7, 20]. The authors of [7] investigated the effects of profile length for an explicit and rating-based elicitation strategy (in the movie recommendation domain) and suggested that the optimal number of ratings is more likely to be 10. Even though in our approach the data is collected implicitly (rather than explicitly), based on the findings of [7] we decided to remove all the users with less than 10 invocations logged to make sure that we would have a minimum of information available to split the data between training and testing sets. After the cleaning up phase, data from 159 users remained for our study.

For the step 3a, we used the first T invocations (under varying values for T) of a given user as the training set and the remaining ones as the testing set. After identifying the possibly relevant entities for a given user (step 3b), we evaluate the quality of the predictions made using two well-established metrics, namely precision and recall (step 3c). Precision measures how often the approach makes an appropriate prediction. In our context, an appropriate prediction is achieved for a given user when our algorithm suggests an entity that is, in fact, covered in the future. Recall measures how many of the operations invoked by a given user are actually predicted by the algorithm. Table 3 shows for example the values of precision and recall achieved with T=10. Due to space limitations, the table shows only an excerpt of the results (the original table contains a total of 63 users). To preserve confidentiality, the data in the table have been anonymized by removing user names and by changing the actual operation names.

We observed the behavior of our approach when using different configurations. We repeated the study using different training set sizes (10, 20, 30, 40, and 50 invocations) and different values for the Jaccard similarity coefficient (0.50, 0.66, and 0.75). The training set size defines the amount of information (in our case, number of invocations) used to create the user profile. The Jaccard similarity coefficient, on the other hand, influences the decision of whether two given users are similar or not. For example, when using Jaccard similarity coefficient equal to 0.50, two users *A* and *B* are considered to be similar if the intersection of the operations invoked by them is at least 50% of the union of the operations invoked by each user individually.

Table 4 shows the overall precision and recall achieved when using different values for the Jaccard similarity coefficient. As we can see, the best results for both precision and recall have been achieved when using similarity equal to 0.75. A possible explanation for this result is the fact that the algorithm is more conservative when making predictions and, consequently, the predictions are more accurate. A trade-off of using high values for the Jaccard similarity coefficient, however, is that less predictions are made. As we can see in Table 4, the amount of users that received recommendations (predictions) varied between 23 and 63, depending on the different sizes of training sets (10, 20, 30, 40, and 50). Thus, it is important to find a balance between the number of predictions made and the accuracy that is appropriate for the environment in which the approach is being applied.

Table 4: Precision and Recall achieved for different values of Jaccard Similarity Coefficient.

Jaccard Similarity Coefficient	User Span	Overall Average Precision	Overall Average Recall
0.50	49 to 100	73%	67%
0.66	38 to 88	78%	68%
0.75	23 to 63	97%	75%

Table 5 shows detailed information about the values of precision and recall achieved for the different training set sizes when using Jaccard similarity coefficient equal to 0.75.

Table 5: Precision and Recall achieved when using training sets with different sizes (similarity = 0.75).

Training Set	# Predictions made	Avg. Precision	Avg. Recall
10 invocations	63	99%	72%
20 invocations	44	98%	77%
30 invocations	36	96%	75%
40 invocations	28	97%	73%
50 invocations	23	96%	77%

Although the reported results are preliminary and cannot be generalized to other systems, they are promising and encourage us to perform further studies.

5. RELATED WORK

In this work we have introduced a coverage metric, the social coverage, for customized test adequacy and selection criteria. The topic of adequacy criteria has been deeply explored by researchers. Since the notion of a test criterion (i.e., the criterion that defines what constitutes an adequate test) was introduced in the 1970's by Goodenough and Gerhart [14, 15], a lot of contributions have been made on the definition of new test criteria [19, 27, 17, 11] and considerable research effort has also been devoted to the comparison of multiple criteria to provide support for the use of one criterion or another [4, 25, 10, 26, 9].

Although many contributions have already been done in the topic of coverage testing, the definition of new adequacy criteria (or the adaptation of the existing ones) has not been able to follow the same pace of the emerging paradigms (e.g., component-based development, service-oriented architecture, and cloud computing). This point was raised, for example, by Bartolini et al. [3]. In this work, the authors argued that traditional test approaches should be revised to deal with service-oriented systems, and proposed a methodology called SOCT (service-oriented coverage testing) that enables the use of practical test adequacy criteria to service-oriented applications.

Our proposed metric was supported by the idea of relative coverage whose definition allows us to have a flexible and context-dependent number of targeted entities that does not necessarily comprise the full set of available entities of a given kind. Relative coverage has also been used in [8]. In this work, the authors proposed an approach in which testable services (services instrumented to provide their clients with coverage information) are provided along with test metadata to help their testers to get a higher coverage. In their approach, coverage is calculated based on a list of operations of the testable service that are actually used by a given client. Differently from our work, in which the list of relevant entities is defined in an automated way based on coverage data collected from similar users, in [8] the list of relevant entities is manually defined by the user.

Various proposals are advocating to leverage information that come from the community of users in a collaborative testing approach [2, 23, 24]. In [2], for example, the authors proposed a test-broker architecture in which all the stakeholders within the composite web service can contribute to improve the testing of the services. Besides supporting the submission, indexing, and querying of test artifacts such as test cases, defect reports and evaluations, the test broker can also provide the services for the test generation, test coordination, and distributed testing services.

In [24] the authors suggest that Collaborative Verification and Validation (CV&V) should be used as the testing paradigm for web service testing instead of Independent Verification and Validation (IV&V). The proposed approach can be used to rank the fault detection capability of test scripts and to establish the oracle for most test inputs.

The authors of [28] proposed an approach in which past failure data of similar neighbors is used to predict the reliability of a given web service. This is very similar to our work as we also use past data of similar users to provide coverage information to a given tester. The main difference when compared to our approach is that, while in [28] the reliability of web services is predicted without providing further support for the testing activities, in our work we attempt to extrapolate the topic of coverage testing by customizing coverage information for a tester based on what operations have been invoked by similar users and by suggesting possibly relevant entities that have not been tried yet.

6. CONCLUSIONS AND FUTURE WORK

In this work we have introduced Social Coverage, a coverage metric that customizes coverage information for a tester based on coverage data collected from similar users. The proposed approach can mitigate the limitations of traditional coverage and can provide meaningful coverage results to testers, guiding them towards increasing test coverage in the areas that are relevant to them. We have hereby defined the approach, implemented an instance of it, and performed a preliminary study to evaluate its potential.

The results achieved in our preliminary investigations suggest that, in similar environments, social coverage can provide a better support to testers than traditional coverage. In our exploratory study, our approach achieved satisfactory values for the precision (ranging from 73% to 97%, depending on the Jaccard similarity coefficient adopted) and recall (ranging from 67% to 75%).

In the future, we plan to extend this work in several ways. First, we plan to adopt different techniques to find similar users (e.g., Pearson Correlation, Euclidean Distance, etc) and to recommend entities. This will allow us to objectively compare the different techniques is terms of the quality of the recommendations achieved, the computational cost, and the actual time required for the similarity calculation. Second, while in this work we studied the capability to predict a user/service coverage based on similar users/services, we plan to investigate whether social coverage can actually improve the testing process cost-effectiveness ratio. Third, we plan to investigate the usefulness of social coverage when considering different adequacy criteria such as statement coverage and branch coverage, for example. Finally, we plan to perform further studies to obtain a better understanding of the cost, performance, and potential benefits of adopting the social coverage adequacy criterion.

7. ACKNOWLEDGMENTS

The authors wish to thank the colleagues of the D4Science project for their collaboration in the exploratory study. Breno Miranda wishes to thank the Brazilian National Council for Scientific and Technological Development (CNPq) for providing his scholarship grant.

8. REFERENCES

[1] P. Andrade, L. Candela, D. Castelli, A. Manzi, and P. Pagano. The D4Science Production Infrastructure. Technical Report 2009-TR-054, Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, 2009.

[2] X. Bai, Y. Wang, G. Dai, W.-T. Tsai, and Y. Chen. A framework for contract-based collaborative verification and validation of web services. In *Proc. of the 10th International Conference on Component-based Software Engineering*, CBSE'07, pages 258–273, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti. Whitening SOA testing. In *Proc. of the 7th joint European software engineering conf. and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 161–170, New York, NY, USA, 2009. ACM.

[4] V. R. Basili and R. W. Selby. Comparing the effectiveness of software testing strategies. *IEEE Trans. Softw. Eng.*, 13(12):1278–1296, Dec. 1987.

[5] A. Bertolino, G. De Angelis, A. Sabetta, and A. Polini. *Trends and Research Issues in SOA Validation*, chapter 4, pages 98–115. IGI Global, 2012.

[6] G. Canfora and M. Di Penta. Testing services and service-centric systems: Challenges and opportunities. *IT Professional*, 8(2):10–17, 2006.

[7] P. Cremonesi, F. Garzotto, and R. Turrin. User effort vs. accuracy in rating-based elicitation. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 27–34, New York, NY, USA, 2012. ACM.

[8] M. Eler, A. Bertolino, and P. Masiero. More testable service compositions by test metadata. In *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*, pages 204–213, 2011.

[9] P. Frankl and S. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. *Software Engineering, IEEE Transactions on*, 19(8):774–787, 1993.

[10] P. G. Frankl and S. N. Weiss. An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria. In *Proceedings of the symposium on Testing, analysis, and verification*, pages 154–164. ACM, 1991.

[11] P. G. Frankl and E. J. Weyuker. An applicable family of data flow testing criteria. *IEEE Trans. Softw. Eng.*, 14(10):1483–1498, Oct. 1988.

[12] gCube Development Team. gCube Website. `https://www.gcube-system.org`, 2008.

[13] P. Good. Robustness of Pearson correlation. *Interstat*, 15(5):1–6, 2009.

[14] J. Goodenough and S. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, SE-1(2):156–173, 1975.

[15] J. Goodenough and S. Gerhart. Toward a theory of testing: Data selection criteria. *Current Trends in Programming Methodology*, 2(2):44–79, 1977.

[16] H. Hemmati and L. Briand. An industrial investigation of similarity measures for model-based test case selection. In *Proc. of the 2010 IEEE 21st Int. Symposium on Software Reliability Engineering*, ISSRE '10, pages 141–150, Washington, DC, USA, 2010. IEEE Computer Society.

[17] B. Hetzel. *The complete guide to software testing.* QED Information Sciences, Inc., Wellesley, MA, USA, 2nd edition, 1988.

[18] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2Nd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '08, pages 208–211, New York, NY, USA, 2008. ACM.

[19] G. J. Myers. *The art of Software Testing.* John Wiley & Sons, Inc., New York, NY, USA, 1979.

[20] A. M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: An information theoretic approach. *SIGKDD Explor. Newsl.*, 10(2):90–100, Dec. 2008.

[21] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[22] H. Sun, Y. Peng, J. Chen, C. Liu, and Y. Sun. A new similarity measure based on adjusted euclidean distance for memory-based collaborative filtering. *Journal of Software (1796217X)*, 6(6), 2011.

[23] W.-T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang, and R. Paul. Testing web services using progressive group testing. In C.-H. Chi and K.-Y. Lam, editors, *Content Computing*, volume 3309 of *LNCS*, pages 314–322. Springer Berlin Heidelberg, 2004.

[24] W.-T. Tsai, Y. Chen, R. Paul, N. Liao, and H. Huang. Cooperative and group testing in verification of dynamic composite web services. In *Computer Software and Applications Conference COMPSAC 2004. Proc. of the 28th Annual International*, volume 2, pages 170–173 vol.2, Sept 2004.

[25] S. N. Weiss. Comparing test data adequacy criteria. *SIGSOFT Softw. Eng. Notes*, 14(6):42–49, Oct. 1989.

[26] E. Weyuker and B. Jeng. Analyzing partition testing strategies. *Software Engineering, IEEE Transactions on*, 17(7):703–711, 1991.

[27] M. Woodward, D. Hedley, and M. Hennell. Experience with path analysis and testing of programs. *Software Engineering, IEEE Transactions on*, SE-6(3):278–286, 1980.

[28] Z. Zheng and M. R. Lyu. Personalized reliability prediction of web services. *ACM Trans. Softw. Eng. Methodol.*, 22(2):12:1–12:25, Mar. 2013.