Using Testing Trace for Automatic User Categorization

J. Jenny Li and David M. Weiss
Avaya Labs Research, 233 Mt. Airy Rd. Basking Ridge NJ 07920
{jjli|weiss}@research.avayalabs.com

Abstract

Testing has always been an indispensable part of software development. With the increasing amount of testing, the volume of data and information generated from testing grows substantially. The question arises on how to take advantage of the testing data, besides traditional coverage and debugging. In this paper, we propose an approach of using test trace data of a software application to its run-time user categorization. It collects test execution trace of programs studied by the software tool, and derives internal metrics of different categories from the trace information. During run time, we look at the user's artifacts as well as the user's behavior to categorize them into predetermined groups and serve them accordingly. Our work in-progress is to apply this method to a software product line, PolyFlow, including a web service that generates, runs, and analyzes test cases of programs under study. One benefit of our method is that it does not require storage of user profiles.

1. Introduction

Success web applications often need to serve a large number of concurrent network users. User profiling is one way to improve such service by grouping and prioritizing user activities, so that similar activities can be served identically to reduce costs, e.g., Google filed a patent on search engine user behavior profiling as given in [1]. User profiles represent individual user interests and preferences in order to satisfy long-term information needs. The goals of user profiling is to tailor information to users' need to avoid information overflow to users, to provide pertinent services to users, and to offer personalized assistance to users.

User profiling and tailored service have several challenges, ranging from profile storage to diversity in service requests. When a web application has a very large number of concurrent network users, providing service tailored to individual user interest is very difficult. User profiling and categorization attempt to solve this problem by putting users into service categories and serve them accordingly.

One major difficulty of user profiling is the storage and retrieval of profiles for a large number of users. Even if sufficient storage is provided, retrieval can slow down service when the number of users gets very large. Keeping individual user profiles requires large storage space and inflicts security risk.

Secondly, most study in the area of user profiling and categorization focuses on helping users to cope with the increasing amount of information available on the Internet. Less attention was paid to the issue of taking advantage of user profiling for improving user assistance and service with reduced costs. For example, when users are detected to be in the same category, they can be put into the same social group to exchange experience and documents, i.e., identifying group similarity and helping each other.

Furthermore, user categorization can be costly. For new applications, it may be difficult to predetermine profile categories. On the other hand, most applications are heavily tested before their release. There are vast amount of testing data that can be used to predetermine user categories, which are not used by the current state of practices.

Our method attempts to resolve the above three problems. We divide up user profiling and categorization technology into three major steps: user category determination, user profile collection and category-based responses. We surveyed existing technologies of the three tasks and compared them to our method.

In some situations, user categorization is required and in others, it can be skipped. When user's service depends on categories, then categorization is required, an example of which is a call center application. When a zip code is entered, the user can be grouped into regions and connected to the regional operator. On the other hand, some applications may not require

categorization. For example, in a one-on-one web application session, the user profile can be collected and the user is served right away without putting into a category. When the user shows up again, his/her profile can be pulled out and the user may not need to reenter information again. Furthermore, for applications with user categorization, there are two types, the dynamic one allowing addition and removal of user categories during run time and the static ones having the user categories decided before the actual execution. Static ones require pre-knowledge of user categories. For example, a call center can have predetermined locations of operators. Our method uses a static pre-categorization method which makes use of trace data of test cases. It derives the range of metrics values from test traces to determine user categories.

Most user profiling [2] collects knowledge of who the users are and what they want as the first step to meet users' needs. For example, a search engine profiler uses key words to predict users' future usage. Our method collects user's run-time application execution trace, other than users' behavior as most user profiling does. Namely, we collect program execution trace to be used for user categorization. Besides, we also collect users' artifacts. Since our user profiles are transient, i.e., discarded as soon as it is categorized, we do not have a storage space issue even with additional information collected.

When determining user responses based on user categories, existing technologies use similarity in word mapping. One example is to use ontology to decide categories of words and their corresponding responses. For example, if someone searches a similar word and from the search gets to a certain web application link, the same link will most likely to be used again by the same user with the same search. The existing response technology uses such information to feed related advertisement to users. Our method matches users' run-time execution trace with the test trace categories. Besides user behavior, we also look at their artifacts. Once a match is found, appropriate service actions, such as setting up a group communication conferences and linking to proper level of documents, etc, can be taken right away, and user profiles can be removed immediately. No storage of user behavior insures that user privacy is intact.

The applicable scope of our method depends on web application categories. For some applications, user profiling can include user artifacts besides user inputs and outputs. Our method focuses on this type of applications. Namely, we treat user artifacts as part of user profiles. Most of software engineering tools such as IDE, testing tools, etc, belong to this category. We will run experimental trials of our approach on a software testing tool.

The rest of the paper is organized as follows. First in section 2, we will describe in detail our test trace based user categorization method. Section 3 presents an on-going case study of applying this method to a web software application with potentially very large number of concurrent users. Section 4 concludes that our test trace based method is feasible for run-time user categorization.

2. Test-Trace-Based User Categorization

Our Test-Trace-Based (TTB) method includes the three user profiling steps. Figure 1 shows the setting and workflow of the TTB method.

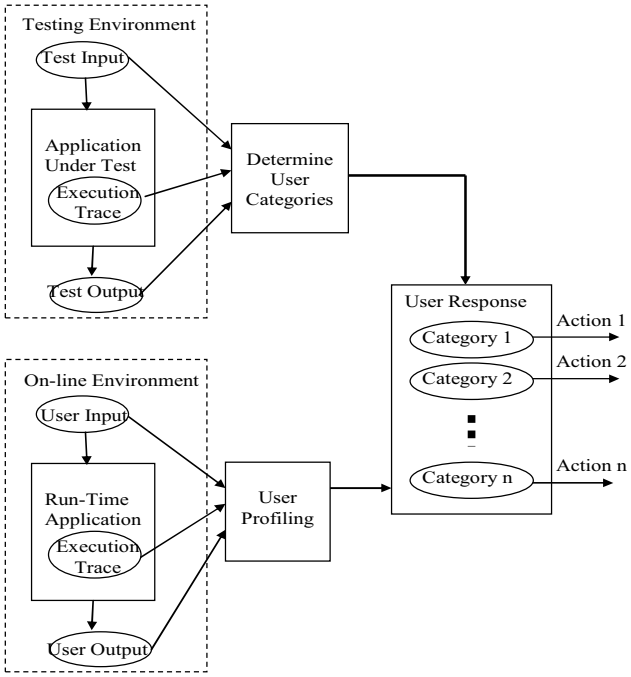

Figure 1. The three tasks of TTB method

Similar to other user profiling methods, our approach includes three major tasks: 1) determining user categories, in which we differentiate by leveraging test traces collected during testing; 2) user profiling to collect both user behavior and metrics of user artifacts; and 3) determining response based on users' profile categories and taking immediate actions so that user profiles can be discarded right away.

The input to user categorization, the dash box on the upper left side of Figure 1, indicates that the TTB method uses test inputs, outputs and execution trace for user categorization. The lower left dash box shows the information collected during run-time user profiling.

Besides execution data, the trace information can include historical metrics of the artifacts, which will be given in details in the following subsections.

2.1. Determining User Categories

TTB method uses test trace to derive user categories. Testing traces are often collected during coverage or debugging type of testing. Coverage testing inserts probes into program under testing either at run time or compilation time. As an example, consider a Java class, "Graph", with defined methods, such as construct, remove, draw, zoom, and etc. Coverage testing of class "Graph" would typically insert probes into each method of the class to record execution trace. The probes often reside at the basic blocks [5] of each method. Suppose the "zoom" method inside "Graph" class has a control flow of basic blocks as shown in Figure 2.

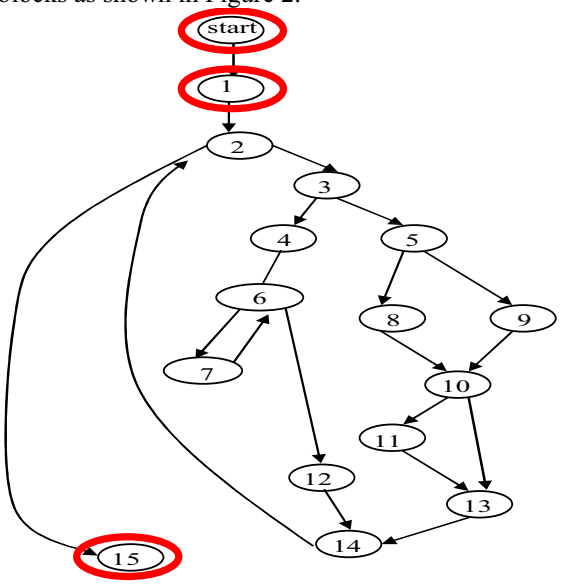

Figure 2. An Example of Tracing Probe Locations

For user categorization, we need to obtain the following information: 1) the sequence of features used and the frequency of their uses; 2) metrics of the program under study by the software tool, such as size, complexity, number of changes and names of the person making changes; and 3) input and output sequences of the tests.

To obtain the above information, the testing trace for the program under testing must first be filtered to include only needed information. The first type of trace is collected by probes residing at the user interface to collect the input and output of the user. The parameter values collected at the starting node of Figure 2. The second type is on invocation of each feature, so that their invocation sequence and frequency can be collected. The probe at node 1 of Figure 2 can provide this information. The third type is inside the program, some selective basic blocks, which provides information of program metrics. Node 15 of Figure can obtain this information.

Besides selecting right basic blocks to get trace information, the third one might require looking into the other supporting software development tools of the program under study. As discussed previously, the focus of this work is the software development tools such as IDE. These tools are often used to analyze the program under study. The analysis results are collected during testing to validate test results, which can also be used to provide the third type of information for deciding user categories. For example, white-box testing tools often collect program complexity metrics and version control tool often collects metrics of program changes. We can make use of such information for user category determination.

Using testing trace of above three types of data, we can then decide the number of categories and their division criteria. The category identification algorithm includes setting a range for each metrics, for example, developer types, maybe as simple as novice, inexperienced, and expert. The inexperienced ones may have code change frequency of say less than 25 a week and expert would have more than 25. Then the criterium of whether the code change number is larger or smaller than 25 can be used to separate users into categories. For the non-metrics factors, such as feature invocation flows. We can use the similarity among the flows to decide categories. For example, users using a feature more than 30 times at the third spot of feature trace can be put into one category.

2.2. Collecting Run-Time User Profiles

Unlike coverage testing where all execution traces are collected, run-time user profiling will only collect needed information to determine user categories. The key different of this step and the previous step is data collection carried out during run-time, which imposes spcae and time constraints. The target web server is instrumented with only the three types of probes needed for user catogarization and traces are collected accordingly. For the example given in Figure 2, only three nodes need to be instrumented with probes, other than all 15 nodes as selected for coverage testing.

Overall, at run time, the TTB method obtains the following information of each user, 1) feature trace of the user, 2) metrics of the user's artifacts, and 3) the user's input behavior. Note that the outputs to the user

is not collected because it is generated after categorization and responses, which is another difference between run-time profiling and testing.

2.3. Determining Responses

With user profiles and pre-known categories, we can decide the user category on the flyer. In addition, the TTB method response determination enables the user to contact other users who are currently available in various groups to get help. Furthermore, the method provides the user with a passive notification that he or she should consider seeking help about certain issues related to the metrics collected. There are other possible responses. For example, a novice may get connected to the same group for appropriate level of documents and connected to a higher group for help.

While knowing who your users are and what they want is the first vital step in meeting their needs, appropriate and prompt response is the most important next step, which is directly visible to users.

3. A Case Study: PolyFlow Testing Web Service

One of our on-going works is to apply the TTB method to a software product line, called PolyFlow [3]. PolyFlow includes several web service products that have potentially very large number of concurrent users. When we were designing the PolyFlow GUI for user assistance, we came up with the idea of applying TTB user profiling. The TTB method was created to address the issue of handling large number of concurrent web users appropriately and promptly.

PolyFlow is a product line of software testing tools, including a minimum capability of executing tests and calculating associated coverage measure [4]. It includes 34 commonalities and 41 variabilities, each variability having an associated parameter of variation that has from 2 to 6 values. An example commonality is that all family members must derive the control flow graph of the program under test and be able to display it for the user. A variability example is the programming language of the system under test, such as Java or C.

Figure 3 shows PolyFlow's setting for using TTB method. It includes 1) a list of features that can be provided to user, 2) user artifacts with data from other applications that can be used, such as version control and error reports, and 3) user's input such as name, identity, location, etc.

The first box on the left of the Figure 3 represents the users that provide login information to the service, including name, file location, etc. The middle box is the GUI on the user side which is instrumented with probes to record the sequence of feature usage. For PolyFlow, example of features includes displaying CFG of the program, getting code coverage, getting performance profile, getting slicing bug localizer results, and automatically generating test cases [5]. The right side box of Figure 3 is the web server that provides the actual feature services on user artifacts, i.e. the program under testing. The artifact links in its version control server and error reporting server.

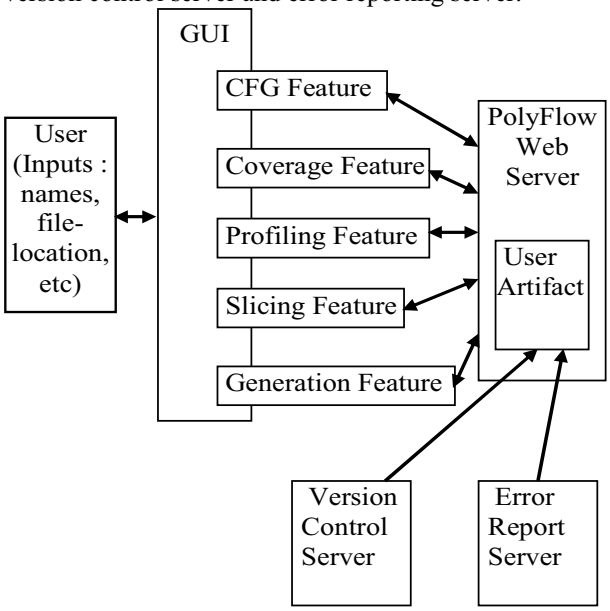

Figure 3. PolyFlow Setting for TTB Method

The version control server and error report server can give additional information for user categorization. For example, version control server can provide change frequency information and error report server gives out the number of error fixes attributed to the user.

For the first user categorization step of the TTB method, test cases of PolyFlow are run on the setting of Figure 3. The test input includes the users' inputs and artifacts. Recording of the test trace will provide a set of data for each test case. For this PolyFlow example, the trace will include the following information, 1) user name; 2) user file location; 3) sequence of feature usage; 4) size and CFG complexity of the program under testing; 5) change frequency of the program, 6) number of error reports associated with the users, and 7) the number of errors fixed by the user.

Figure 4 shows the format of a test trace of PolyFlow products. It includes three segments: header, trace body and results. Figure 4 also shows that test trace provides more information than the seven types of information we need because testing trace is also used for features such as coverage measurement. On the

other hand, it does not provide the last two types of information, which can be fetched from other servers.

As discussed in the previous section, not all trace information is need for user categorization. We use a trace filter to select feature-related trace. "User name" is the first item in a trace header (input to a class method) and can be derived easily. "User file location" on the other hand does not appear in the trace directly. We can use project name information to infer file locations stored in the project file. This case shows that we are using test artifacts besides behavior and trace.

User Name Project Name	Header
Method-begin …… Node-in-method …… Method-begin …… Last-method-end	Execution trace
Code complexity CFG complexity ……	Result

Figure 3. Format of testing trace

Derivation of "feature usage" information is quite different from the first two types of information. It needs to be filtered out of the test execution trace of the entire program. As discussed previously, the test trace records execution of every line or every control flow node (basic block) of a program. The filter needs to point to the entry of a feature-invoking method.

The abstraction of artifact complexity metrics is again different. These numbers are the internal variables of the applications, i.e. they are variable values recorded at some point of a test execution that contains useful metrics information. These numbers are only available at testing time and may be transient during actual run-time usage.

The version control server connected with PolyFlow will provide data on change frequency of the program under testing. Frequent changes often indicate less mature code. The bug tracking server provides historic error information of the program, of which the bug fix numbers of the user can be extracted.

With the seven types of information from testing, we will put users into three categories, novice, experienced and expert. The division criterion will come from actual data of the testing trace. Once the categories and their division criteria are set, they will be stored at GUI for user profiling and response.

The PolyFlow user-profiling step records the seven types of information during the run-time. The recorder also resides at the GUI module. Run-time recording will require fetching data related to the user artifact from the web server. It also needs to record some internal metrics of the users and discard them as soon the user category is decided.

User response determination is also carried out by the GUI module. When it detects a misusage or usage scenario that was never tested in test trace of a novice user type, it will set a red flag on the tool manual. The red flag is not intrusive because if the user chooses to ignore the flag, no action will be taken. Only when the user click on the flag, proper actions such as linking user to the same user category to get help will be taken to give the user appropriate assistance.

Traditional method of user support by giving an on-line manual does not address the issue of servicing different categories of users. The TTB method will enable PolyFlow to provide prompt user assistance based on user categories.

As mentioned previously, the TTB method does not require storage of user profiles. This characteristic is particularly useful for PolyFlow because of its product line design that allows its number of features to grow.

4. Concluding Observations

This paper presents a practical usage of testing trace for user profiling to improve web service tailored to user needs. It describes the proposed Test Trace Based method and its potential application to a software product line with web service products.

The differentiator of TTB method includes 1) usage of existing testing traces; 2) usage of user artifacts besides regular user profiles; 3) transient user profiles; 4) non-intrusive user notifications; and 5) tailored service such as linking users' of similar profiles.

Application of the TTB method to our PolyFlow product line is ongoing. More relevant results and data will be presented in the future.

5. References

[1] http://www.searchenginejournal.com/google-patent-organic-results-ranked-by-user-profiling/2448.

[2] http://www.freepatentsonline.com/y2007/0260517.html.

[3] D.M. Weiss, J. Jenny Li, H. Slye, T. Dinh-Trong, and H. Sun, "Decision-Model-Based Code Generation for SPLE", Proc. Of 12th International Software Product Line Conference, pp129-138, Sept, 2008

[4] Qian Yang, J. Jenny Li, and David M. Weiss, "A Survey of Coverage-Based Testing Tools", The Computer Journal 2007; doi: 10.1093/comjnl/bxm021

[5] J. Jenny Li, D. M. Weiss, and Y. Lee, "Coverage-Guided Prioritized Test Generation", Journal of Information and Software Technology, V. 48, I12, pp1187-1198, Dec. 2006.