Technical Disclosure Commons

Defensive Publications Series

November 2020

Critical User Journey Test Coverage

Carlos Arguelles

Tylor Sampson

Joseph Kubik

Erez Bibi

Critical User Journey Test Coverage

ABSTRACT

In the space of software testing, making sure that all critical user scenarios or journeys (CUJs) in a product have been tested is important. A CUJ is essentially a series of steps in the product taken by users of the product to achieve a desired outcome. Ensuring CUJ coverage during testing is a difficult problem that often requires subject matter experts to identify critical scenarios and expensive manual efforts to track coverage. This disclosure describes automatic extraction of CUJs based on analysis of test logs and production logs, e.g., traffic logs at a website. Data mining techniques are applied to merge various sources of data, deduce critical user journeys, and prioritize them during testing. Code release can be gated based on test coverage of a threshold proportion of the identified CUJs.

KEYWORDS

- Unit testing
- Integration testing
- Software testing
- Gap analysis
- User journey
- User scenario
- Traffic log
- Data mining

BACKGROUND

In the space of software testing, making sure that all critical user scenarios or journeys (CUJs) in a product have been tested is important. A CUJ is essentially a series of steps in the product taken by users of the product to achieve a desired outcome. For example, for an email service, a CUJ with the outcome of sending an email may include the user pointing the browser to the service provider's website, clicking on a "Compose" button, selecting a recipient, typing a subject, typing the body of the mail, and clicking on a "Send" button.

Ensuring CUJ coverage during testing is a difficult problem that often requires subject matter experts to identify critical scenarios and expensive manual efforts to track coverage. Current processes are time consuming, require a high degree of expertise, prone to errors, and can become out of date, as customer behavior and product features change. The impact of not testing all CUJs is that code changes may inadvertently break critical user scenarios, go untested for a long time, and be broken in production, causing customer pain and loss of customer trust in the product.

Tools are available to keep track of line and branch code coverage that record which lines of code or branches of logic in the code were executed by tests. These are generally effective for unit testing, but fail to capture customer journeys for integration testing. The objective of integration testing is not restricted to determining whether a line of code was executed, but evaluating whether a customer scenario was executed. Thus, identification of an exhaustive list of CUJs is essential for effective integration testing. Existing solutions for identification of CUJs involve annotating production and test traffic. As a result, these solutions require a high degree of expertise and involve manual effort.

DESCRIPTION

This disclosure describes techniques that involve analysis of traffic logs, with user permission, to automatically extract critical user journeys (CUJs) from patterns that are found frequently in the logs. The application of the described techniques can eliminate the need for expensive subject matter experts to manually list the expected scenarios, and ensure that the identified scenarios are up to date and representative of the requirements of real-world customers.

Data mining techniques are applied to merge various sources of data, deduce critical user journeys, and prioritize them for testing. Data mining is performed on traffic logs (suitable processed to ensure compliance with user permissions for use of such data) from the production environment to create actual CUJs. Traffic logs from test environments and test cases are analyzed to produce a list of the tested CUJs. The list of actual CUJs from production and the list of tested CUJs can be compared to display to the testing engineer the coverage achieved during testing.

Fig. 1 illustrates an example process for the extraction of CUJs from data sources such as application trace and production/test logs.

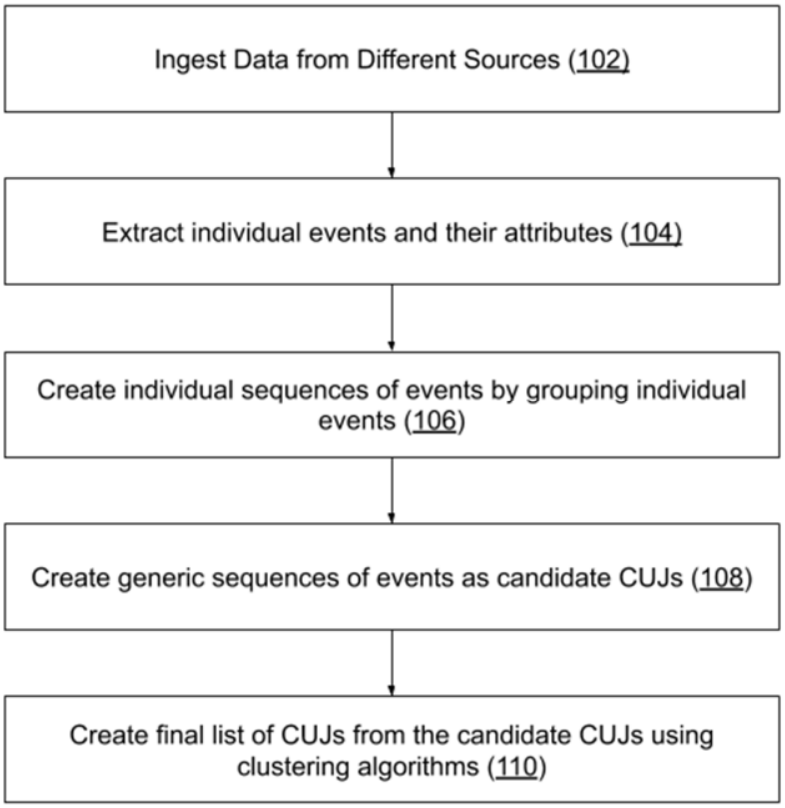

Fig. 1: Steps in extraction of CUJs from traffic logs

Data from a number of different data sources such as traffic logs from a production or test environment or application traces are ingested (100). Data can be received in streaming mode or batch mode. Different sources of data (processed to remove user identifiable information) are merged based on heuristics such as user id, trace id for a transaction, or timing information into a single source of truth.

The ingested data are parsed to extract individual events (102) such as API calls, remote procedure calls (RPC calls), HTTP transactions, etc. and attributes of the event (e.g., input/output values for an API call, prior state, headers for HTTP connections). For example, for a retail e-commerce site, an individual event may be "Customer X viewed item Y at time t1", another example may be "Customer X purchased item Y at time t2." Related individual events are grouped into individual sequences of events that logically belong with each other as part of a customer workflow (104).

In the above example of the retail site, events that occur within a certain time of each other and are related to the same customer are merged, e.g., to obtain a single "Customer X viewed item Y and bought it" workflow for the example above. Individual sequences of events that share the same application programming interface (API) sequence (execute the same code portions) are combined into a generic sequence of events (106). In the example of the retail site, this may include a workflow of API calls of "a customer views an item and purchases the item." All individual occurrences of particular sequences of API calls are combined to deduce critical user journeys. For the retail site example, the generic sequence would contain different variations of "customer X_i views item Y_i and purchased it."

The candidate CUJs obtained based on the sequence of events and several individual instances of these CUJs are then subjected to a clustering algorithm. Clustering algorithms such

as k-means or mean-shift clustering are applied to a set of many individual instances of the CUJ that have different data to identify instances that share commonalities. Such analysis also reveals CUJs that need to be broken into multiple CUJs (108), e.g., when the source events have different attributes. For example, the sequence “customer views item x, customer purchases item x” may be mapped to multiple different CUJs, e.g., one for purchasing electronics and another for purchasing food.

The techniques described with reference to Fig. 1 are applied on production data to generate CUJs. A similar mechanism is then applied on the test traffic to generate test CUJs. As shown in Fig. 2, these two sets can be compared to generate a report on the coverage of the testing process and also to highlight gaps in CUJ coverage. The coverage of the test can be utilized for gating a code release, e.g., it may be required that testing of each code version meet a minimum coverage threshold prior to release of code to the production environment.

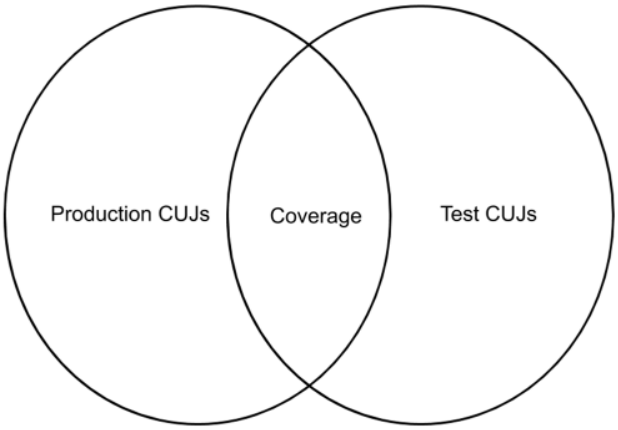

Fig. 2: Evaluating coverage of the testing process

While creating the individual sequence of events or generic sequence of events, the order in which events occur is important. Sequences of events may also include superfluous events that do not matter in the CUJ. For example, “customer views item 1, customer views item 2,

customer purchases item 1" and "customer views item 1, customer purchases item 1" may be treated as the same CUJ - in this case, the fact that customer viewed item 2 is superfluous, and is therefore disregarded.

While grouping workflows into CUJs, an account is maintained of the number (or proportion) of instances of each CUJ that are seen in production. This information is leveraged to determine the relative importance of individual user journeys. By default, CUJs that are executed more often have a higher priority for testing. However, testing can be customized to allow users to assign priority to different CUJs based on attributes of the CUJ. Further, an adaptive algorithm can be used to generate CUJs with similar weight as production CUJs which can ensure that CUJs are covered with meaningful attributes.

The mapping of CUJs from testing to those from production can be utilized to identify tests that had the most coverage as well tests that had the most important CUJ coverage. This data can be utilized to modify the execution order of tests to execute the most critical tests first, or to break up a large test suite into small sub-suites with different priorities. The CUJs can also be sorted by importance when being subjected to human auditing to prioritize important CUJs, which can reduce the cost of human audits. The prioritization of CUJs can also enable creation of a smaller suite of tests that executes faster and covers the most important CUJs, and that can be executed earlier and/or more frequently in the code development process than the full suite of tests.

While the foregoing description refers primarily to online applications/ web services, the techniques can also be applied to testing mobile or desktop applications, e.g., to generate CUJs based on user interactions with the application, from users that permit use of such data (after suitable processing to remove identifiable information).

CONCLUSION

This disclosure describes automatic extraction of CUJs based on analysis of test logs and production logs, e.g., traffic logs at a website. Data mining techniques are applied to merge various sources of data, deduce critical user journeys, and prioritize them during testing. Code release can be gated based on test coverage of a threshold proportion of the identified CUJs.