WHITE PAPER

A.I. for Risk Based Testing and Production Driven Test Coverage

Discover the current A.I. capabilities and the future of not just A.I., but also the future of software testing. What does the future of true A.I. automation hold for testing?

Introduction

A.I. can be used in many ways in testing. This white paper will discuss one specific approach that is driven by analyzing large amounts of production data that describes actual usage by real users through the data taken behind them and the metadata that describes in various ways what they've actually done. This is done on top of more traditional sources of test coverage analyzing requirements, exploratory testing, etc. Analyzing this production data by actual users is a very good risk-based approach to testing highly complex systems based on what users actually do instead of all of the theoretical permutations of what a system may need to support. For example, a heuristic process may help determine the core of the vast majority of business processes, which will let you focus on which business cases actually get used.

Before we can discuss Artificial Intelligence (A.I.) and our success in including A.I. in the testing process, we need to discuss pattern recognition, since pattern recognition is an important component of A.I. Imagine you're a child, and your parents show you pictures of animals and ask what sounds they make. Before stimulating the knee-jerk reactions of cow-goes-moo and dog-goes-woof, a complex pattern recognition process takes place, where anything from a photo of an animal's head to a line drawing of its whole body gets interpreted and mapped to that animal. A.I. at its simplest is a machine's ability to recognize patterns. We take for granted what we recognize easily, and marvel at what is beyond us, but you need to realize that this is simply a sliding scale in terms of computer programming, or applied learning. In 1984, a man studied supposedly random patterns by walking through stop-motion on a VCR, and used what he learned to walk away with $110,237 on a game show called Press Your Luck.

Now, let's go beyond pattern recognition to smart reactions to those recognized patterns. You're playing tic-tac-toe for your very first time. You start and place your very first character, an X, in the middle of a side. You soon learn that this is the wrong way to start, and that anyone

A.I. Today in Software Testing

Is software testing capable of employing A.I. approaches, or will human software testers soon be bested by computers as well? Software testing already includes A.I. strategies, based on pattern recognition which feeds into the next calculated process.

Let's begin by looking at some A.I. testing accomplishments. Imagine aggregating logfiles from multiple sources into a repository that is scanned for warnings which may not even display on-screen. The preceding actions in the logs can help determine the buggier apps and areas of code requiring closer analysis. This Business Intelligence approach already exists. A.I. in testing can also be used in the following ways:

- Explore user experience, by analyzing text from social media feeds (sentiment analysis) to spot feedback trends about what has already been released
- Cluster similar bugs together by data visualization heat mapping, for easier attack by Development (via the Pareto Approach, theorizing that bugs like to nest together)
- Reduce test cases that can be determined to be unnecessary before execution?
- Predict if specific follow-up DevOps sprints require specific tests cases to be run, vs. being omitted because there's no chance that the problem got addressed yet.

Qualitest Group has also tackled A.I. approaches to software testing, by analyzing groups of bugs and bug metadata to identify significant patterns about the underlying software systems and teams that develop them. Here, similar bugs are clustered together, and the code in those areas is studied for irregularities, such as outdated code or lack of compliance to coding standards, resulting in bottom-up and higher-level root cause analysis. Qualitest Group and other vendors have also developed combinatorial analysis methods to logically reduce test cases, acting in a decision support capacity to a test architect professional who drives the process.

Risk-Based Testing

Suppose the user must fill out a complex submission form. Logic says to test all possible combinations. But what if that total is too high, perhaps infinite? Which test cases are most important and most likely to encounter failure, and how do you limit the count? You may have a number or date field, but choose to limit testing to its boundary and near-boundary values.

Imagine a form with 8 dropdowns, each with 8 possible values. If testing a combo takes a minute, testing every combo will take just shy of 32 years. However, pair-wise testing (which tests all combos of each parameter pairing), would only take around an hour and a quarter.

Lastly, let's look at state transition tables, where we limit our test cases based on a focus on state changes, like a phone dialer's 3 states: accept more digits (default), execute a call, or go to an error message table. Focusing on the transition paths between these three states, our total test case set resembles multiples of 3 instead of a phone number's powers of ten.

How A.I. can help the 5 basic sources of test coverage

Let's look deeper at the 5 basic sources of test coverage, defining each one, then look at what A.I. can theoretically do to help us, toolwise.

- Reviewing specifications tell us what a program should do and how it should work. A.I.'s pattern matching helps us eliminate unneeded "too close" test cases by seeing which ones are too similar. As we mentioned earlier, this may mean focusing on boundary value analysis (edge cases, literally), emphasizing state transition, or ensuring all-pairs testing.
- Exploration testing session logs, via pattern recognition of verbose logging, seek activity patterns of specific warnings tracking to specific user actions, modules, forms, etc.
- Known product issues, once analyzed, can have A.I. cluster similar bugs through pattern recognition, suggesting likely duplicates. Bugs from automated test cases can be auto-run on previous builds to find the causal build to help pinpoint root cause code changes.
- Discussions with knowledgeable personnel (product owners, developers or Marketing, etc.) may determine code danger areas. White box-driven test design targets the actual revised code, hunting for specific code level problems. Factors may include the coder, change

A.I. pattern searching) show how much time each type of user spends in different program areas on different actions. Early focus on these heatmapped areas concentrate attention where the most user time is spent.

Implementation of Verbose Logging

We've mentioned user logfiles, but must emphasize their need to have useful content designed for analysis, as James Bach discussed in http://www.satisfice.com/blog/archives/401. Logfiles function like an airplane's black box, recording exactly what happened leading up to a crash (or any other moment in time), helpful for future precision analysis to properly recreate what happened. You can leverage a better solution of what went wrong by examining that easy-to-parse data with pattern-matching as a basis for strategy and innovation, such as which apps are failing, and what events and warnings, etc. immediately precede the failures.

Leveraging Client QA through the use of A.I.

One of Qualitest Group's client's web-based application is used to generate reports, using 20 different applications access nearly 200 different data sources. Some reports are generated by paid subscribers while others are free, bringing in no income. Customers typically select a filter (time, durations, data source, server type, company, department, etc.) and run a saved report or customize a new report based on that metadata. Some customers have reported some data problems in ad hoc reports. The client believes the problems are due to the processing of this data once a report request is issued, not the quality of the collected metadata in the database.

The client's customers generate about 4 million reports a year. The number of possible paths through the software to generate these ad hoc reports is presently unknown, but we estimate that manual testing time would exceed 35 man years. So, how does one choose an optimal test case subset for testing today, and for regression testing later? We were brought in to help narrow the focus. What criteria can be used for tossing out the vast majority of test cases? Maybe A.I. can recognize production behavior patterns.

We began by using production driven test coverage and combinatorial analysis. This cutting edge approach to test design and test management is based on online analysis of data and metadata as indicators of functional usage gaps between different environments, i.e. test vs. production. This approach leverages COTS visualization software packages such as Weave or Tableau to visually represent data and metadata patterns as a test designer guide to identify gaps in test coverage. Our method offers 3 significant benefits:

- Online indication of gaps between how real life users use a product vs. how it is being tested. This identifies gaps (hidden patterns) that could not be found otherwise.
- Risk-based testing led by metadata factors such as indicators of coverage on functionality that is revenue generating vs. not, that is key user focused vs. not, and so on. This results in improved test coverage, profit/revenue and customer satisfaction.
- A method to define a subset test coverage (usable for regression or smoke testing) that is comprehensive and sufficient in cases of an infinite amount of functionality and combos which are impossible or prohibitively expensive to cover.
- All 3 benefits emphasize test cases focused more on real user, not avoided, behavior.

Once we understood the overall structure (DB schema, table joins and relationships, how information flowed between tables, and how it flowed between the metadata and Sybase), we began our pattern-hunting exploration. We filtered and analyzed the hits or counts in every category, the traffic, hours of activity, permission levels, data sources targeted, app servers and DB servers utilized for the same over a month. Next came comparisons with test data, using the same analysis approach for an apples to apples comparison.

Weave successfully found the patterns we sought, analyzing more axes at once than people can. We displayed how concentration patterns differed between production and QA's efforts. Production environment usage metrics showed the daily, weekly and monthly reports types as well as filters used. The study included report count analysis for different modules representing different business names, usage of app servers and DB servers, different frequencies or data sources, access roles for permissions and security, and suppression use, showing how emphasis strayed between Production and QA for each area. This analysis showed QA environment testing gaps clearly, as well as performance concerns during heavy usage that were outside the initial project scope but easily visible from the observed data, and of interest to the customer.

As we proceed, A.I. strategies will help map known bugs to gaps in coverage. Meanwhile, the human effort will have engineers investigate data problem sources giving consideration to Bad Calculations, Incorrect Data Segmentation, ETL Issues, Data Quality Issues, and Display/GUI Issues among other possible causes. The test engineers will ultimately select several workflows to analyze in depth and will run pilot tests of various possible test automation solutions.

Through use of A.I. techniques, we at Qualitest Group can now make predictions about data usage for 3, 6, 9, 12 months down the line by looking at past data patterns. Any information gained through A.I. can only help strengthen the communication between Business and QA, helping to further goal creation, the overall QA process, and information transfer. And that's a goal that any non-artificial intelligence would be happy with.