ISSN: 0974-8571 Vol. 13 No. 2 December, 2021

International Journal of Computational Intelligence in Control

Natural Query Based Retrieval of Executable Testcases To Understand Currency in Live Software Environments

S Reine De Reanzi, P Ranjith Jeba Thangiah
reanzi@gmail.com, prjt@live.com

Date of Submission: 13th March 2021 Revised: 6th April 2021 Accepted: 13th April 2021

How to Cite: **Reanzi, S. and Thangiah, P., 2021. Natural Query Based Retrieval of Executable Testcases To Understand Currency in Live Software Environments.** ***International Journal of Computational Intelligence in Control, 13(2).***

***Abstrac*t-Existing software products undergo transformations to adopt the latest technologies in order to remain relevant. It is challenging to keep track of the currency feature behaviors of such an application under transformation (AUT) and ensure it does not impact the existing customer behaviour. In this paper, we report a method implemented as a proof-of-concept prototype called testFabric, an approach for posing a natural language query to retrieve a test that can be executed on AUT, to find out currency feature behavior. The approach explains codeless automation of tests which is used to retrieve (i) an appropriate fully executable test (ii) test libraries to be selected, then sequenced into a fully executable test. With the use of wit.ai (ML platform by Facebook), it is found that the results are promising, where a natural language query can retrieve an executable test.**

Keywords : test retrieval, test selection, test sequence, microservices, SaaS, query, codeless automation, ML

INTRODUCTION

The need fora on-demand, reliable customer onboarding and retention drives software companies to adopt newer architecture by adopting technologies like microservices to enable shorter time to market at scale. Microservice is designed in a manner consistent with the single responsibility principle [3].

For a microservice API, the signature [18] is published up front. So, during technology transformation, it must be ensured that the impact on existing scenarios is close to zero, especially when the application under transformation (AUT) is delivered via the cloud. This paper discusses an approach called testFabric that addresses this problem.

OBJECTIVE

To retrieve and Execute a Test in Response to a Given Query Using an ML Platform. The purpose of this study is to build a testFabric prototype and the structure of study is as follows.

(i) Build a tool for creating robust, reusable test libraries that can be used in a plug-and-play manner. Develop codeless engineering to assemble selected test libraries into a test. (ii) Use the application-based domain-modeling approach [1] to create feature keywords and tags to map functional behaviors and their variants to tests and libraries. (iii) Enumerate the possible ways a user can query for a scenario and build a training and validation dataset. (iv) Use classification algorithms to create a test suite of classified data models. (v) Prototype testFabric with plug and play machine learning platform to predict correct tests and test libraries to resolve user queries that are written in natural language. Retrieve the resulting tests for execution against the AUT for a given user query. (vi) Perform primary and secondary evaluation and assessment of results with respect to quantitative and qualitative parameters. (vii) Discuss future work.

BACKGROUND

An existing monolithic software-as-a-service (SaaS) product [10] that adopts a microservice architecture is considered in this study. Here, the functional behaviors of the workflows are already known. With live customers on production, the product mix, volume, pattern, usage, typical issues, expectations, infrastructure, capacity, throughput, etc., are known and baselined.

The tests are modularized in a way that enables continuous integration, verification, deployment and delivery (CI/CV/CD) [16], with the required coverage [14], using combinatorial test design [15]. Once services have been deployed, a set of functional tests are required to verify the behavior of a given function.

The existence of numerous services makes it challenging to determine the intended behavior of a given system [2]. Is the

behavior the same after the new changes?The testing and investigation effort necessary for a feature to cover all of its dependencies is exponential [4]. This is where ML can be leveraged.

REVIEW OF LITERATURE

Microservices is the present-day approach for application architecture,where the application is developed as components, where each component is a full, but a midget application that is focused on realizing a single business task. The task is implemented end to end, application programming interface, the database and may include user interface as well. The services can mostly run independently, likewise development can happen independently. On the similar lines, the test automation can happen independently. One of the best practices of developing a microservice API is to publish its signature [18] first, for the intended business task. Refer Single responsibility principle [3]. This helps the development teams, both producing and consuming, to work independently. This enables the automated tests to be developed in parallel as the signature is usually agreed upfront between the development teams. There are several frameworks like pact[1], which are used for writing contract tests [8], where the tests are written for validating the meta data of the service. They are primarily written by the consumer teams as white box tests and are sometimes shipped with the code.

Once services are deployed, we need a set of functional tests to verify the behaviour of the given function at hand. There are various tools like rest-assured[2] etc., to write automated tests for the services. There will be numerous services for a product/platform. Hence it becomes very difficult to quantify coverage, identify the missing coverage, assess if a test is present for a given functionality or to just run a test to find out if the intended behaviour of the functionality on the given system.

A great deal of work is also being carried out on the testing of microservices using AI. Work has been done on AI-based XPath identification [11], screen matching [4], UI element matching [4], UI label pattern matching [5], domain modeling [9], change-based automatic test generation [13], spidering AI for test generation and anomaly detection [7], AI-based code synthesis and test generation [20], coverage-oriented AI-based suites [17], etc. There are also some AI-based test generation tools that can generate test methods based on given code. However, to the authors' knowledge, there has been no previous research on the use of AI to retrieve and execute tests in response to user queries expressed in natural language. In this study, such an approach has been tested and validated on a SaaS product [10].

CODELESS TEST FOR A MICROSERVICE ARCHITECTURE

Using Customized Uniform Resource Locators (CURLs) as the Uniform Resource Identifiers (URIs) of the necessary services, along with the required parameters and tokens, a test to be run on an application can be formulated to run with corresponding test data.

A basic UI was built to create a visual mechanism to facilitate codeless automation and execution.The UI presents three options for creating a library: setup, action or validation. Based on the results obtained with validation libraries, a test is marked as either pass or fail.

Each library is named in accordance with predefined standards to enable the identification of the intention of the library (Fig.1).Hence, reusable codeless tests and test libraries are achieved. These are now ready to be used by an ML system.

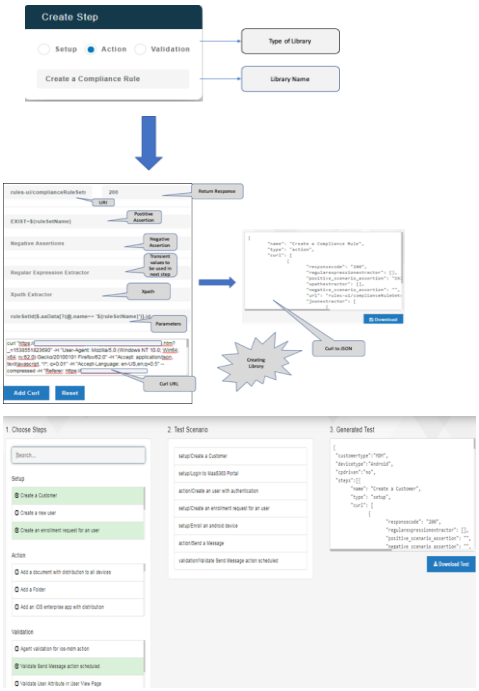


Figure. 1 Creation of a codeless test library

The following (Fig.2) is the architecture of the framework.

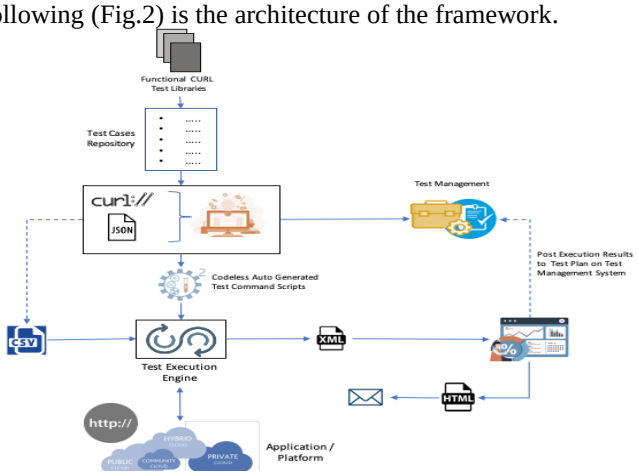


Figure. 2 Architecture of the framework

1 https://docs.pact.io

2 http://rest-assured.io

COMPARISONS OF THE SOLUTION

The tests were rewritten to optimize the test coverage using combinatorial test design [15]. This resulted in less tests but more coverage. This activity was done before curl framework. Hence the GUI based automation was done for optimized suite as well. The following charts (Fig.4) represent the comparison of the results. There is a huge savings on time and considerably less failures (Fig.3).

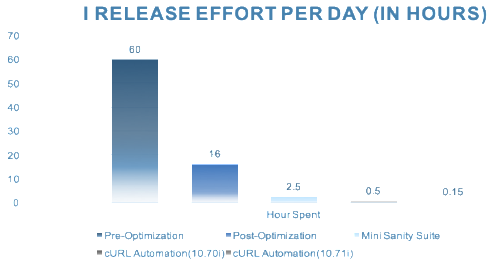


Figure. 3 Reduction in execution time

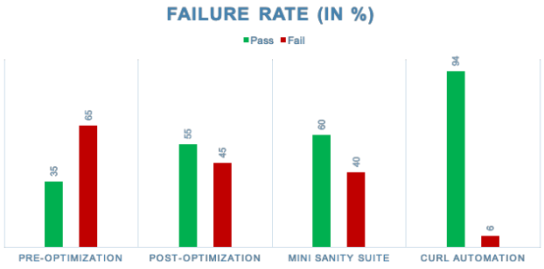


Figure. 4 Reduction in failures & false positives

Since it was designed to run on any environment and can be triggered at will, by anyone, the SaaS – ops team was given a switch to trigger after the infra-patching activity of production is completed. This eliminated the need for testers to be around. Clearly, this framework turned out to be winner with reusable codeless tests and test libraries. These are now ready to be used by ML system.

NEED FOR AI BASED TEST QUERYING & EXECUTION

Along with the humongous refactoring activity, the new product feature development and enhancements delivered along with it to meet customer and market expectations made it complex. So when the underlying strucure is being changed it is challenging to ensure the existing behaviour for the customers is unchanged.

One of the issues that we face time and again is that, there were redundant queries that came up from different stakeholders apart from engineering; like sales, support and product managers were around, how does it work now?, With the new changes is this behaviour same now?, is this scenario addressed, this was a bespoke one for the customer, is it retained and does it work the same way?, how many customers are impacted? etc. Each time there is a query, the tester digs the required test from the suite and executes them on the current code to find out the result. If a test is not available, the setup and the tests are created then executed.

The testing effort of a feature with all its dependencies covered is exponential [4]. And finding a sliver out of it, tosimultaneously query its behaviour with execution is complex and also time, effort and resource consuming. Hence the need for AI

MATHEMATICAL REPRESENTATION OF TEST LIBRARIES AND TEST CASES

The set of test libraries is denoted by T_L and consists of elements from T_{Ls}, T_{La}, and T_{Lv}, where 's' denotes setup libraries, 'a' denotes action libraries and 'v' denotes validation libraries.

$$T_L = \{\{T_{Lsi}\}, \{T_{Laj}\}, \{T_{Lvk}\}\} \text{ for all i, j, k} \quad (1)$$

Therefore, test T_C for a scenario is represented as an ordered sequence.

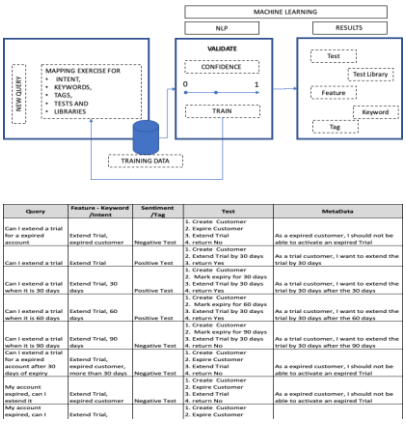
$$T_C = (\{T_{Lsi}\}, \{T_{Laj}\}, \{T_{Lvk}\}) \quad \text{for any i, j, k} \quad (2)$$

CONSTRUCTION OF THE AI-BASED TESTFABRIC

To address the issues mentioned in Section III, the idea is to take a natural language query as an input from a user and find (i) a whole executable test (structured) or, (ii) if a whole test is not available, the necessary test libraries (unstructured) to be assembled into an executable test to determine the real-time behavior of the application of interest. This enables the retrieval of context-based tests [6] in response to a given query.

I. Data Model For Processing Query

The next step of creating the testFabric is to devise a data model to represent natural language queries, feature keywords, tags, tests and libraries.



Query	Feature – Keyword /Intent	Sentiment /Tag	Test	MetaData
Can I extend a trial for a expired account	Extend Trial, expired customer	Negative Test	1. Create Customer 2. Expire Customer 3. Extend Trial 4. return No	As a expired customer, I should not be able to activate an expired Trial
Can I extend a trial	Extend Trial	Positive Test	1. Create Customer 2. Extend Trial by 30 days 3. return Yes	As a trial customer, I want to extend the trial by 30 days
Can I extend a trial when it is 30 days	Extend Trial, 30 days	Positive Test	1. Create Customer 2. Mark expiry for 30 days 3. Extend Trial by 30 days 4. return Yes	As a trial customer, I want to extend the trial by 30 days after the 30 days
Can I extend a trial when it is 60 days	Extend Trial, 60 days	Positive Test	1. Create Customer 2. Mark expiry for 60 days 3. Extend Trial by 30 days 4. return Yes	As a trial customer, I want to extend the trial by 30 days after the 60 days
Can I extend a trial when it is 90 days	Extend Trial, 90 days	Negative Test	1. Create Customer 2. Mark expiry for 90 days 3. Extend Trial by 30 days 4. return No	As a trial customer, I want to extend the trial by 30 days after the 90 days
Can I extend a trial for a expired account after 30 days of expiry	Extend Trial, expired customer, more than 90 days	Negative Test	1. Create Customer 2. Expire Customer 3. Extend Trial 4. return No	As a expired customer, I should not be able to activate an expired Trial
My account expired, can I extend it	Extend Trial, expired customer	Negative Test	1. Create Customer 2. Expire Customer 3. Extend Trial 4. return No	As a expired customer, I should not be able to activate an expired Trial
My account expired, can I	Extend Trial,		1. Create Customer 2. Expire Customer	

Figure. 5 Query-Feature-Keyword-Tag-Test mapping (snippet from the training set)

The queries and the feature keywords and tags are first formulated based on domain expertise for all workflows and scenarios. As a result, a comprehensive domain dataset is obtained. The structured tests are appropriately tagged with

the relevant keywords, and fine-grained test-keyword-tag mappings are constructed. The dataset is then split inthe ratio 80:20 for training and validation purposes. (Fig 5) is presented to better illustrate the data model.

II. ML Training: Mapping Tests and Test Libraries

Two-fold training is performed, as specified below:
(i) ML Training: Tests, Libraries, and Queries
ML Training Input Set: query; **ML Training Expected Output**: test, test library
(ii) Tagging for Context (test-keyword-tag mapping)
ML Training Input Set: query, keyword-tag mapping, test; **ML Training Expected Output**: test, test library
The following (Fig. 6) demonstrates the training scenario

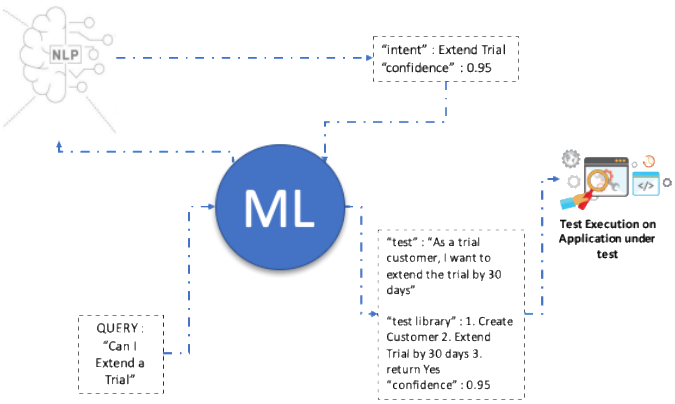


Figure. 6 Example of a scenario being trained

(Fig.7) is the real-time training step on wit.ai, notice how the entities are prompted after a few records of training, which have the correct context.

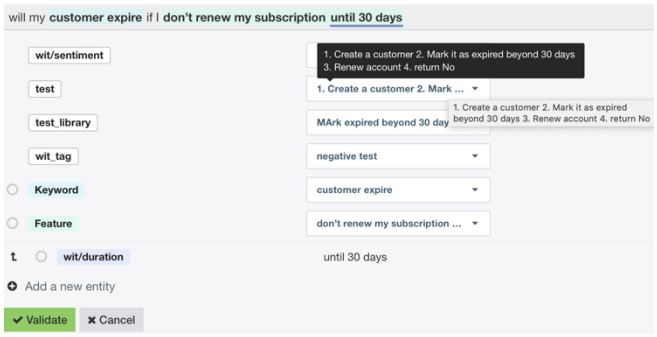


Figure. 7 Training for mapping of Query, feature, keyword, library, test, tag

The complete training data can be constructed as a json file and imported to train the ML system. The snapshot of the training data is presented in (Fig.9).

III. Proof of Concept Using wit.ai (an ML plug and play Platform from Facebook) [19]

The wit.ai platform, uses ML-driven feeding of data, for the system to learn on its own. The training set from (Fig 5) is used to train, and the intent is only to imbue the model with an intelligence quotient [12].

The following (Fig. 8) portrays on how the wit.ai platform uses the query, intent, trait, key words etc. to train the model. The training data is used to train the platform.

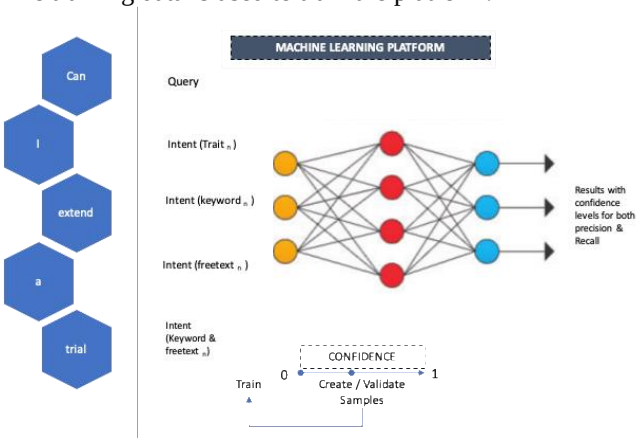


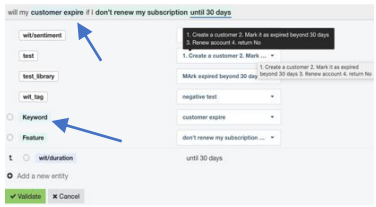
Figure. 8wit.ai's messaging platform

Figure. 9 Training dataset in json format

After training, use the validation data to extract a whole executable test or a set of test libraries, along with a confidence level. If the confidence level is satisfactory, the test retrieved can be used for execution. This form of supervised learning assists in fine-tuning the accuracy of the results. It also contributes to increased coverage.
The complete training data can be constructed as a JSON file and imported to train the ML system. The search strategies that are employed in wit.ai are trait, keyword and free-text strategies. The desired intent is learnt by the algorithms.

IV. Initial Qualitative Evaluation – Training Dataset

After a few records have been used for training, it is observed that related keywords, features and tag values depend on the context. (Fig. 10) shows a view of real-time retrieval of test queried using curl command on wit.ai and the output with confidence value. Note that the suggested/predicted keywords and test from already-trained values are highlighted in green. The context is validated to be correct. This proves that ML-based test retrieval is possible.



curl\-H'Authorization:BearerJSJITLC76M2Y35HMITBIFA5GWFS7UED2' \
'https://api.wit.ai/message?v=20190908&q=can%20I%20extend%20a%20trial%20for%20a%20expired%20account'

{"_text":"can I extend a trial for a expired account",
"entities":{
"Feature":[{"entities":{"Keyword":[{"confidence":0.99451042911744,"value":"extend a trial","type":"value"}]},"confidence":**0.99862216335192**,"value":"trial account deleted","type":"value"}],
"test_library":[{"confidence":**0.9617406193555**,"value":"expire customer"}],
"test":[{"metadata":"As a expired customer, I should not be able to activate an expired Trial","confidence":**0.99103634586862**,"value":"1. Create Customer 2. Expire Customer 3. Extend Trial 4. return No"}]},"msg_id":"1lZr6IEXZaDPCdMEU"}

[a] Response for the Curl from Table II

Figure. 10 Training for the mapping of query, context & CURL output

After the training is completed, it is tested with a validation dataset. The test is retrieved with confidence level mostly above .80, where the correctness of context is verified to be true. The ML platform retrieves the data in the form of json depicted in code snippet below.

{"_text": "my customer expire if I don't renew subscription until 30 days",
"entities": {
"Keyword": [
{
"confidence": 0.9919678572793,
"value": "customer expire",
"type": "value"
}
],
"Feature": [
{
"entities": {
"Keyword": [
{
"confidence": 1,
"value": "renew a subscription",
"type": "value"
}
],
"duration": [
{
"confidence": 1,
"value": 30,
"day": 30,
"type": "value",
"unit": "day",
"normalized": {
"value": 2592000,
"unit": "second"
}
}
]
},
"confidence": 0.98668558735246,
"value": "don't renew my subscription until 30 days",
"type": "value"
}
],
"test": [
{
"confidence": 0.99825574225576,
"value": "1. Create a customer 2. Mark it as expired beyond 30 days 3. Renew account 4. return No"
}
],
"wit_tag": [
{
"confidence": 0.99895124479424,
"value": "negative test"
}
],
"sentiment": [
{
"confidence": 0.7144933981909,
"value": "neutral"
}
]
},
"msg_id": "1GhkuOHsCXh2g2KpZ"
}

Hence the tests and test libraries can be retrieved via API or Curl, to be executed on the test environment

V. Qualitative Evaluation – Validation Dataset for Tests and Libraries

The ML training process is repeated until the best mapping is obtained. Once training is complete, the results are tested on a validation dataset. The tests and libraries are retrieved with confidence levels above .80.

QUANTITATIVE & QUALITATIVE ASSESSMENT OF TEST AND TEST LIBRARY SELECTION

The quality of the mapping can be viewed using a confidence chart for precision and recall [19]. The results are given below after training for 100 records of data. The results also show that the more records we use to train, the confidence levels are

closer to accurate. In (Fig.11) & (Fig.12), notice the test is at .75 confidence level

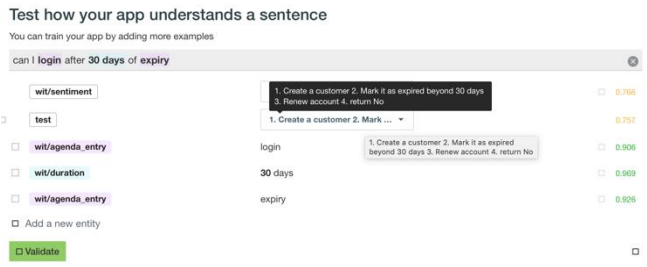


Figure. 11 Training for a usecase query

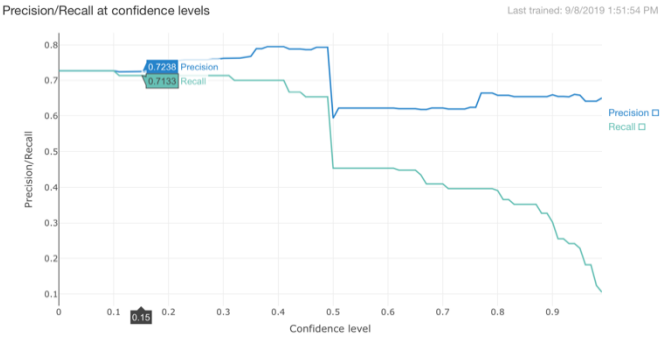


Figure. 12 Precision & recall chart for the trained entities

After more training, we could see that it is possible to get an increased confidence level, in this case the same query has resulted in confidence level .98 (Fig.13). Notice that it has started prompting test library as well at a confidence level of .98 (Fig.14).

The domain correctness from the results show that it is possible to train the usecases to retrieve the required tests or test libraries to be executed against the application under test.

Notice that the (Fig.12) snapshot is taken at 1:51pm on 8thSep2019 and after more records of training again, a snapshot (Fig.14) is taken at 7:33pm on 8thSep2019.

It is clear that with more context to the machine learning system, the confidence level of retrieval of the test is close to accurate at .98.

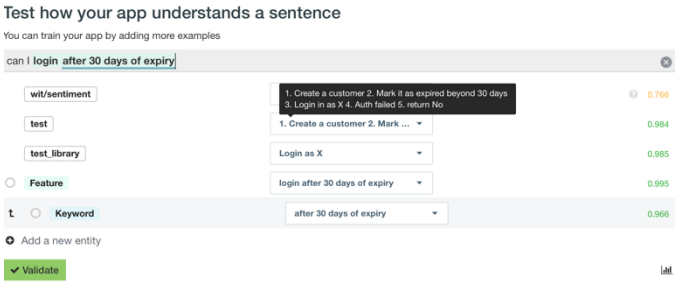


Figure. 13 Improvement in Confidence level after more records of training data

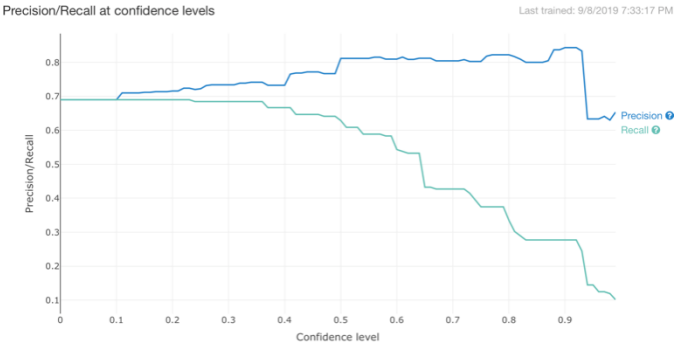


Figure. 14 Improvement in confidence levels of precision and recall

The comparison of (Fig.12), (Fig.14) shows that it is possible to train the AI system to select tests, test libraries that are already automated and can be executed on application under tests, to get real-time status of the behaviour for the use-cases.

CONCLUSION AND FUTURE WORK

The results clearly show that an AI-based testFabric is possible. To make the testFabric more effective, the same approach can be extended to contract tests [8], which are used for the verification of API signature metadata [8]. Furthermore, the testFabric can be enhanced by providing the context of a persona. This can help the testFabric retrieve tests for a specific persona (support, customer, product manager, etc.).

The testFabric can also be enhanced to display appropriate messages when no answers are found and to prompt users to provide suggestions or feedback to fine-tune the suite.There can be many benefits of the proposed testFabric - (i) Reduction in time spent on investigation to understand the existing behaviour. (ii) Reduction in time to detection of failure. (iv) Increased coverage in first attempt. (iii) Execution of corner cases at a lower cost with fewer false positives. (vi) Optimal quality assurance head count with minimal tacit knowledge dependency, real-time documentation, etc.

ACKNOWLEDGMENT

The authors would like to thank IBM MaaS360, a SaaS application, where the experiment was conducted. Thanks to Vinod Rajiah, IBM for his help in implementation.

REFERENCES

[1] Abramov, J., Sturm, A. (2010) Supporting layered architecture specifications: a domain modeling approach. https://doi.org/10.1007/978-3-642-13051-9_17

[2] Bider, Ilia., Halpin, Terry., Krogstie, John., Nurcan, Selmin., Proper, Erik., Schmidt, Rainer., Uko, Roland. (eds) Enterprise, business-process and information systems modeling. Springer, Berlin, Heidelberg, pp 195-207. ISBN: 978-3-642-13051-9

[3] Ampatzoglou, A., Tsintzira, AA., Arvanitou, EM., Chatzigeorgiou, A., Stamelos, I., Moga, A., Heb, R., Matei, O., Tsiridis, N., Kehagias, D. (2019) Applying the single responsibility principle in industry: modularity benefits and trade-offs. In: Proceedings of the evaluation and assessment on software engineering. Association for Computing

Machinery, Copenhagen, Denmark, pp 347-352. https://doi.org/10.1145/3319008.3320125
[4] Arbon, J. (2017) AI for software testing. In: Pacific NW software quality conference. PNSQC
[5] Baek, YM., Bae, DH. (2016) Automated model-based Android GUI testing using multi-level GUI comparison criteria. In: Proceedings of the 31st IEEE/ACM international conference on automated software engineering. Association for Computing Machinery, Singapore, pp 238–249
[6] Batarseh, FA., Gonzalez, AJ., Knauf, R. (2013) Context-assisted test cases reduction for cloud validation. In: Brézillon P, Blackburn P, Dapoigny R (eds) Modeling and using context. Springer, Berlin, Heidelberg, pp 288-301
[7] Colantonio, J. 5 great ways to use AI in your test automation. https://techbeacon.com/app-dev-testing/how-ai-changing-test-automation-5-examples
[8] Dai, G., Bai, X., Wang, Y., Dai, F. (2007) Contract-based testing for web services. In: 31st annual international computer software and applications conference (COMPSAC 2007). IEEE, Beijing, China, pp 517-526
[9] Gao, J., Tao, C., Jie, D., Lu, S. (2019) Invited paper: what is AI software testing? and why. In: 2019 IEEE international conference on service-oriented system engineering (SOSE). IEEE, San Francisco, CA, pp 27-2709
[10] IBM security MaaS360 with Watson https://www.ibm.com/in-en/security/mobile/maas360
[11] Leotta, M., Stocco, A., Ricca, F., Tonella, P. (2016) Robula+: an algorithm for generating robust XPath locators for web testing. Journal of Software: Evolution and Process 28:177-204 https://doi.org/10.1002/smr.1771
[12] Liu, F., Shi, Y., Liu, Y. (2017) Intelligence quotient and intelligence grade of artificial intelligence. Annals of Data Science 4:179-191 https://doi.org/10.1007/s40745-017-0109-0
[13] Merrill, P. Top trends: 5 ways AI will change software testing. https://techbeacon.com/app-dev-testing/5-ways-ai-will-change-software-testing
[14] Qualitest A.I. for risk based testing and production driven test coverage. https://www.qualitestgroup.com/white-papers/risk-based-testing-production-driven-test-coverage/
[15] Route S (2017) Test optimization using combinatorial test design: real-world experience in deployment of combinatorial testing at scale. In: 2017 IEEE international conference on software testing, verification and validation workshops (ICSTW). IEEE, Tokyo, Japan, pp 278-279
[16] Shahin, M., Ali, Babar M., Zhu, L. (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access 5:3909-3943 https://doi.org/10.1109/access.2017.2685629
[17] Wikipedia Comparison of GUI testing tools. https://en.wikipedia.org/wiki/Comparison_of_GUI_testing_tools
[18] Wilde, E., Pautasso, C. (2011) REST: from research to practice. Springer, New York, NY
[19] wit.ai Recipes for apps you can talk to. https://wit.ai/docs/recipes#which-confidence-threshold-should-i-use
[20] Yen, IL., Bastani, FB., Mohamed, F., Ma, H., Linn, J. (2002) Application of AI planning techniques to automated code synthesis and testing. In: Proceedings of the 14th IEEE international conference on tools with artificial intelligence, 2002 (ICTAI 2002). IEEE, Washington, DC, pp 131-137