Test Case Design and Test Case Prioritization using Machine Learning

Mayank Mohan Sharma, Akshat Agrawal, B. Suresh Kumar

Abstract**: Designing and prioritizing test cases is a very tedious task. Given all the advancements in the world of software testing, on any given day engineers spend several man-hours to identify all possible testing scenarios and preconditions attached with it. Test engineers then use the scenarios and preconditions to write multiple test cases. Every test case has a template skeleton to follow - expected results, actual results, priority, test suite category classification (regression, sanity, smoke, integration, etc.), and the respective software (i.e. version, build, release etc.) that has to be tested. Until now there have been efforts to make test case designing simpler by providing software test engineers with tools and processes. But these tools and processes still needs considerable amount of manual intervention in terms of understanding the requirements, analyzing the quality risks and documentation of all possible test scenarios in order to ensure a high quality software delivery. Man-hours spent on test case design and test case prioritization is directly proportional to the cost involved in building software. Our goal was to make sure that the manual intervention in test case design and test case prioritization is reduced to minimum without imposing any software quality risks. So, that the cost to ship and build software is reduced. With this paper we are presenting a solution to this problem. Our goal here was to use machine learning [5] to do automated test case prioritization and creation of test cases for software. In order to achieve this goal we used supervised machine learning [6] approach based on K-Nearest Neighbor classification model for test case design and test case prioritization [4]. On experimenting with other linear and non-linear classifiers we learnt that they did not prove to be as accurate as K-Nearest Neighbor. Our method of machine learning based automated test case design and test case prioritization can be used by any software development organization to reduce it's software development cost and time taken to ship software to their respective consumers. This aims to benefit the software development industry as a whole.

Keywords **: Test Case Design, Test Case Prioritization, Machine Learning, Supervised Learning, K-Nearest Neighbor Classifier, Software Development, Software Testing.**

I. INTRODUCTION

As the software development shifts more towards being highly agile, the demand to ship code faster is at an all time high, particularly with the adoption of mobile and cloud based infrastructure technology services. One important driver is the fierce competition from other competing technology companies. Most of the development teams are adopting continuous integration [2] and continuous delivery [7]. Every organization is focused on building better, smarter and reliable software applications with customers in mind. With prime focus on mobile applications, organizations still maintain capability for the users to interact with the provided respective services via web and application programming interfaces. Technology companies follow a growth mindset wanting to engage more and more users on all available channels. In order to meet these expectations in a fast paced agile methodology a software test engineer has to work with higher efficiency. They should become capable enough in using the latest tools and technologies available at their disposal to catch defects at the earliest possible stages of development. The cost [17] of fixing the defect increases exponentially; later a defect is found in the software greater is the cost to fix it. The cost of defect found in production environment is far more than to fix it at an early stage in testing environments. In order to find bugs as early as possible the software application has to be tested for all possible test scenarios, conditions, pre-conditions and permutations & combinations of the test data.

So how can a software test engineer keep track of all these hundreds of combinations along with applying those set of test cases to test the software application successfully. The most efficient way to do this has been to create a suite of all the possible test cases to test respective software application, feature, or functionality. But designing of test cases has always been a manual and time-consuming task. Test case execution and automation is much more interesting compared to test case design. Reason being, with test design one has to think and articulate the steps you would follow in order to test certain functionality of the software under test. On the other hand, in test automation or test case execution, a test engineer feels a little more driven as exploring, finding and looking out for hidden defects seems to be an engaging and interesting work. However this hypothesis is not entirely true and it depends on individuals' interests and preferences.

Absence of test design documentation is bound to create gaps while testing certain test scenarios, without any sort of track and follow mechanism. It is extremely difficult memorizing and applying all the numerous possible test cases. Acknowledging and accepting the importance of test case design, finding a solution for the drawbacks in the design process is extremely crucial. It is well known fact that any manual repetitive job is inefficient, unreliable, tedious and time [1] consuming task. All of those factors are immense drawbacks where a need for reliable automation is much needed. Due to rapid changes in functional development under agile [15] methodology, test engineer spends a ton of time writing and re-writing test cases.

Revised Manuscript Received on October 15, 2019.

Mayank Mohan Sharna, Principal Software Test Lead, Zillow Inc, San Francisco, USA. Email: mayank.mohan.sharma@gmail.com

Akshat Agrawal, Computer Science, Amity University Haryana, Gurgaon, India. Email: akshatag20@gmail.com

***B. Suresh Kumar**, Computer Science, Amity University Rajasthan, Jaipur, India. Email: sureshkumarbillakurthi@gmail.com

In order to solve this problem we focused on automating test case designing process. We devised and tested our design engine which is based on parser and machine learning [6], [17] logic to automatically prioritize, categorize and generate test cases for non-complex User Interface (UI) elements of mobile and web applications.

II. APPROACH

A. Selection of Data

We used a test suite of written test cases, which were created for testing user interface of an application. The test suite that we started with was designed to conventionally follow different flows in order to test the application. These test flows traversed through different user interfaces to perform certain actions on the application under test i.e. website, mobile application. For example: Login page with 3 user interface elements - user id text field, password text field and login button uses these elements to login with valid test user credentials. With a different flow two of these elements (User Id and Login button) and a new element i.e. reset password link or forgot password link can be used to test a different flow.

In order to derive a machine learning [6] based mechanism that can predict test cases along with their priority [3] and test suite category [20], we needed to have a dataset, which could be used to build our machine-learning model. For testing a flow in the application it is necessary to determine the quality of the overall functioning of the respective system. However the pages in the test flow change, based on the functionality we are testing. Following a test flow based test cases reference is not suitable to provide relation between a web page, its user interface elements, test case priority [13] and test case category [20] in a way that could help build a predictive model. Additionally, test flow based test design provides the steps to help determine whether the software application works as expected under given conditions for a functionality involving transition of web pages or flow of data etc.

Below is an example of a traditional test case based test design.

Table -I Traditional Test case design flow

Login Flow Test Case Design								
Category	Test Case Name	Priority	Steps to follow	Expected Result	Actual Result	Pass/Fail	Executed By	Test Environment
Sanity	Enter User Id	1	1. Enter website URL and launch the website. 2. Wait for the login page to load 3. Check the label and spelling of the User Id field. 4. Enter correct User Id	1. Website page should load properly. 2. Label of User Id field should be "Enter User Id". 3. Correct user id should be accepted by user id field validators.	1. Page loaded successfully. 2. Label ofuser field has "Enter User Id" label. 3. Correct User id got accepted with no errors.	Pass	Test Engineer: John Doe	Staging
Sanity	Enter Password	1	1. Check the label and spelling of the password field. 2. Enter correct password	Correct password should be accepted by field validators.	Correct Password got accepted with no errors.	Pass	Test Engineer: John Doe	Staging
Sanity	Click Login Button	1	1. Login button should be enabled if user id and password is entered	Login button should get enabled once valid user id and password are entered.	Login button is enabled and when it is clicked it opens home page successfully.	Pass	Test Engineer: John Doe	Staging
Regression	Click Forgot Password	2	1. On the login page enter user id and click Forgot Password	Forgot Password link should be present and spelled correctly.	Clicking Forgot Password opens Password Retrieval Page	Pass	Test Engineer: John Doe	Staging

Below is the test case design [19] based on our approach to derive data for our predictive machine-learning model

Table-II Non-Traditional - Dataset Requirement Based Test Case Design Structure

Login Page Test Design					
User Interface Element	Test Cases	Priority	Cateogry	Number of Test Cases Priority	Number of Test Cases Category
User ID Text Input Field	1. Enter Space 2. Enter Null value 3. Enter only alphabets 4. Enter email address 5. Enter only numbers 6. Enter special characters 7. Enter alphanumeric characters 8. Enter a right username 9. Enter wrong username 10. Exceed character limit	1. P3 2. P3 3. P2 4. P2 5. P2 6. P2 7. P2 8. P1 9. P1 10. P3	1. Regression 2. Regression 3. Regression 4. Regression 5. Regression 6. Regression 7. Regression 8. Sanity 9. Sanity 10. Regression	P1: 2 P2: 5 P3: 3	Regression: 8 Sanity: 2

If we had used an existing test suite that is based on test flow based test case design methodology it would have been difficult to parse and manually quantify test cases for specific user interfaces. Instead we created a new test suite manually first by taking the traditional test suite based on test flows as reference. In the new test suite, we created test cases for a particular web page, which basically were test cases for web page's respective user interface elements. For example: A login page test had 3 user interface elements i.e. login button, user id field and a password field. We wrote tests for each button and respective field. We used individual website pages as our source of test cases related data. Our intention is to use this dataset of test cases to predict a model that provides us with test cases to be applied to a new page.

We wish to know the priority [3] of those tests along with the category they should be placed under. We started with an approach to look at each page of the website as a standalone application. Next we recorded the following parameters for the test cases of each page like User Interface elements present on the page, priorities [3] of test cases for that page for each user interface element, number of test cases for that page for each user interface element, time taken to execute these test cases per user interface via test automation, test cases per test case category per user interface element i.e. regression [11]/sanity/smoke/integration etc. Below is an example of how table looks for one page –

Table-III Test Case Parameters Mapping Diagram Example 1:

Login Page						
User Interface Element	Test Cases Count	High Priority TCs	Medium Priority TCs	Low Priority TCs	Test Case Category	Button Description (Not included in Table)
Button	2	2	0	0	Sanity	Login Button
Text Input Field	8	3	5	0	Sanity	User Id
Text Input Field	8	3	5	0	Sanity	Password
Hyperlink	1	0	0	1	Regression	Forgot Password

Table-IV Test Case Parameters Mapping Diagram Example 2:

Data Entry Page						
User Interface Element	Test Cases Count	High Priority TCs	Medium Priority TCs	Low Priority TCs	Test Case Category	Button Description (Not included in Table)
Drop Down	2	1	0	1	Sanity	Form Type Menu
Check Mark	2	1	1	0	Sanity	Select relevant form fields
Text Input Field	8	3	5	0	Sanity	Input form data
Hyperlink	1	0	0	1	Regression	Help
Toggle	2	0	1	1	Regression	Spell Check On/Off

This table clearly establishes relation between different parameters obtained for a test case. We further wanted to use this type of test case design methodology to find a pattern between positioning of certain user interface elements together and the impact of that positioning on the different test cases based parameters. For example: if the login button and two text input fields for user id and password are found on a page together then the test cases has high priority and the test cases category is sanity and regression [11] both. In the same way if this combination of user interface elements occur on any other page like a form submitting page which has 3 input text fields and a submit button. Then again the test case priority is high and the test case category falls under regression [11] and sanity both. Humans can easily understand that whenever data input occurs and it is sent over to some backend service using a button tap action this test case is generally of high priority. This kind of pattern recognition between the aforementioned parameters was crucial for our machine-learning model to work. If we would have take a test flow based approach then it was very difficult to find this sort of pattern in it. As the prime focus in that approach is based on testing a specific functionality to work and therefore using a test flow based approach to derive a base model was not feasible.

We used this set of data in to evaluate different classification [21] models. In order to extract this data from the non-conventional test case design format we created a parser that parsed the excel sheets of the test cases. The parser we used in this case was written in Python [12] [18]. We used this parser to extract our dataset and then we stored this derived dataset from historic test cases in NoSQL database, which is easy to access and maintain. NoSQL database was chosen because we could easily interact with our dataset using NoSQL database API. Once the dataset was derived and stored in database. We needed to find a mechanism to extract user interface elements so that those could be used as input to predict test cases.

B. Derivation of data

In order to get the user interface elements for the application we wish to predict and create test cases. We had two choices to start with either we could have waited for the user interface to be complete or business requirement documents or flow documents to have description of the user interface elements. We decided to test both of these approaches out to find a solution that could be applied to any type of application i.e. mobile, web or any standalone application. We first tried out the approach to create a user interface page parser that could first generate an element tree of all the user interface elements present on a particular user interface page. Then we could use these extracted user interface elements as an input for our test case design [19] prediction. But the challenge we faced in this approach was that the same parser could not have been used for all types of user interfaces i.e. for mobile, web and standalone application. As for each of them the way to extract the element tree and parsing were unique. A user interface element parser for mobile could not extract user interface elements for web and in the same way a parser for standalone application could be used for mobile or web. Building this kind of parser itself would have been a time consuming task.

So we tried to work on the other approach, which was to use the business requirement documents to find the user interface elements that would be used for the respective pages for a mobile, web or standalone application. This approach seemed to be very straightforward and easy to follow. The only work that we did was to go through the business requirements document and create a table for the user interface elements present on each page of the application. In future if this solution has to be standardized then we can make sure that the business requirement documents are of the desired format. So that it is programmatically possible to parse through these documents and extract the user interface elements. When the product managers or business consultants prepare the business requirements documents the document could be made in such a way that the user interface elements on each page could be easily parsed from it. We suggest the below given format for the same which we adopted for our parser.

Once we obtain what all user interface elements are present on different pages of the application we create a table for the same. The table we created for few of the user elements is given below:

Table-V Table with extracted user interface elements for a page

Page	User Element 1	User Element 2	User Element 3
Login	Button	Text Field	Text Field
Form	Text Field	Drop Down Menu	Toggle

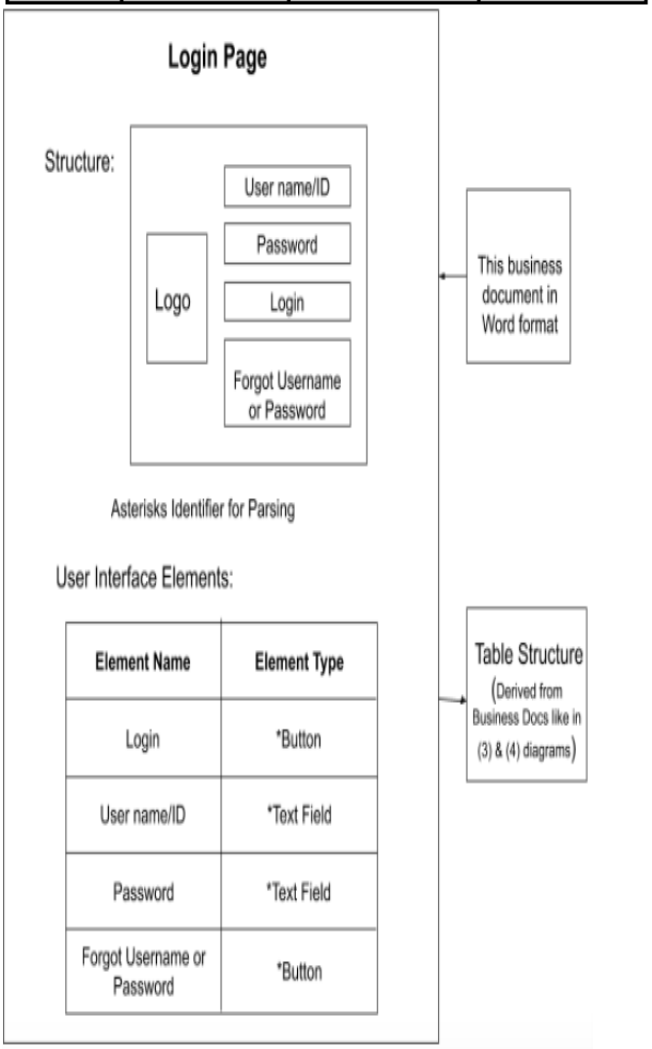

Fig.1 Format for Businees requirement document

III. APPLICATION MODEL GENERATION

A. Approach one-Element Prioritization

We now have dataset for the machine learning [6] to learn from and another set of data that includes user interface elements for which it has to predict test cases and prioritization [4]. Our next step is to evaluate machine-learning algorithms [16] to find which one of the different classifications [21] models would best fit our requirement to provide test case prediction and prioritization [4]. Our primary goal is to predict the number of test cases with their respective priorities for a particular user element that is present on a respective page. Once we get these predicted values we can then manually map the relevant test cases based on our historic test cases and generate a whole new set of test cases. After many failed attempts we finally cracked the code.

In the first test dataset we mapped the test cases count in which a respective user element was mentioned to the respective count of priority-based test cases. But this did not prove suitable to provide us with a mechanism to predict test cases for user elements present on certain user interface pages. As we could not enter number of prioritized test cases or count of test cases as input to predictive model to predict test cases for a new page with same or similar user [10] elements. So this data set was a failed attempt.

Table-VI Login page

Login Page				
Test Cases Count	High Priority TCs	Medium Priority TCs		Low Priority TCs
2	2	0		0
8	3	5		0
8	3	5		0
1	0	0		1

Data Entry Page				
Test Cases Count	High Priority TCs	Medium Priority TCs		Low Priority TCs
2	1	0		1
2	1	1		0
8	3	5		0
1	0	0		1
2	0	1		1

In the second dataset we tried to use the count of prioritized test cases for respective user elements present on a page.

Table-VII Data Set format for each page

Dataset Format (For each page)								
User Interface Elements	Button	Text Field	Toggle	Check Box	Drop Down List	Menu	Hyperlink	Image
High Priority	2	3	2	2	2	2	2	1
Medium Priority	2	2	1	1	5	5	1	1
Low Priority	3	3	3	3	3	3	1	1

This dataset helped us understand the relation between user interface elements and number of test cases based on priorities. On the pages where we did not have any particular user interface element we marked its test cases as zero. This data set was derived for each respective page of the web site and then fed into our machine learning evaluation model. After trying the above approach we identified that this dataset also cannot predict test cases for a new user interface page.

However this first approach of using User Elements, Test Case Priorities and Test Cases Count did not work for us. Our biggest learning was the stick with core fundamental of the design behind an application. The key to success for our system is its strength to determine the workflows and its accuracy in predicting the test cases for any given page based on the relationships learnt based on the historical data [2]. For this particular reason we had to spend time identifying a different approach.

B. Approach two - User interface element count + User interface page mapping

After realizing the failure in the above two different approaches of data set for the prediction model we came up with a mapping of data set that should work best. Following is about the data set we used:

As input for our machine-learning model we provide count of element on a web page. With this we determine the relation of all different types of elements on the web page in consideration. Knowing the different types of elements and the relationship between them helps us determine a pattern of user workflows. These workflows eventually increase the match ratio for the new web pages.

In the below table we are showing how different UI elements are present on different pages, giving an example mapping of pages to respective element types.

Table-VIII Detail of Elements

Page/Element Count	Element 1	Element 2	Element 3	Element 4	Element 5	Element 6
Login	Text Field	Text Field	Button	Check box	Button	Hyper link
Reset Password	Text Field	Button	Button	-	-	-
Home	Hyperlink	Hyperlink	Hyperlink	Hyperlink	Image	Image

Using the above mapping table we enter the count of each element type for each page to create mapping between page and element types.

Table-IX Information of Button type

Page/ Element Type	Button	Textbox	Link	Radio Button	Check box	Drop down	Image
Login	2	2	1	0	1	0	0
Reset Password	2	1	0	0	0	0	0
Home	0	0	4	0	0	4	2

Each page has different count and set of UI elements so by mapping each type of UI element to a particular page we expect to derive a relation which can be used in future to make our test case predictions.

Manner in which a relationship between the above 2 tables is generated can be described as below -

Once a new page is entered in to the system as historical test cases, we identify the different types of elements on the page and extract the count of each element on respective pages.

Using this as input data for our prediction algorithm, we determine the closest resemblance page. Now as we know the closest resembling page we look into its historical test cases [14]. And due to the similarity and close resemblance of the pages, we can now use the historical tests cases as a starting point for the test cases design of the new page. As the number and type of UI elements are largely similar on both the pages t is seen that flows and priority of the test cases also remains similar [10].

For example if we found that the Submit Page (Historical) is similar to the Contact Us page (New) then we find that most of the test cases of high priority like test to fill and submit the information. Test to check the validations for text fields; Test to check for positioning of UI elements etc. remains identical. So when we tested this approach the results derived from the machine-learning model also indicated that the predictions were good and test case design [19] and prioritization [4] was largely precise.

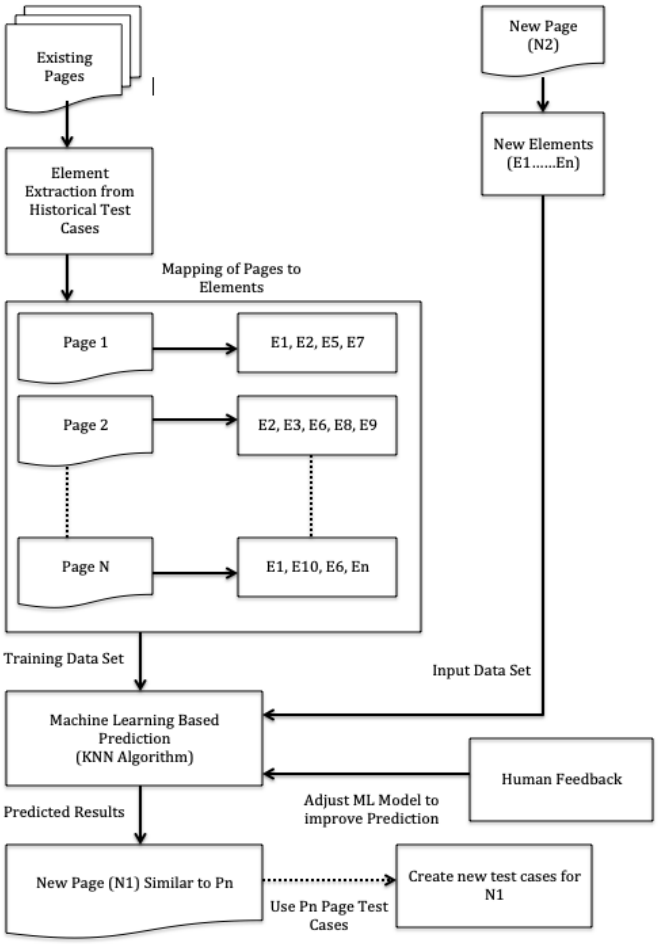

Fig.2 the process flow for our Test Case Design [19] Prediction and Prioritization

As explained in section II we select data i.e. historical test cases [9]. After data selection as described in section 2.2 we derive training data sets in desired format and create mapping of pages to its respective elements. This training data set is used to evaluate classification [21] based machine learning models. The new page for which we wish to predict test case design [19] and test case prioritization is also mapped to its respective elements. This data set becomes the input data set for the machine learning models evaluation process.

In order to test whether our test data is good enough to give any precise predictions we used mapping of all the pages for a website to their respective UI elements. Pages used for test dataset was from About, Careers, Cart and Contact pages. The UI elements that we mapped were button, toggle, hyperlink, and image. This mapped relation based data set was then divided into validation and training sets. Validation data set was a train test split with test size of 0.20. When this data set was validated we used Logistic Regression (LR), Linear Discriminant Analysis (LDA), K-Nearest Neighbor Classifier (KNN), Decision Tree Classifier (CART), Gaussian Naive Bayes (NB) and Support Vector Machine (SVM) algorithms to evaluate the training. It was found that consistently the K-Nearest Neighbor Classifier algorithm came out to be the most precise. The test result obtained shows comparison of results obtained for respective algorithms used:

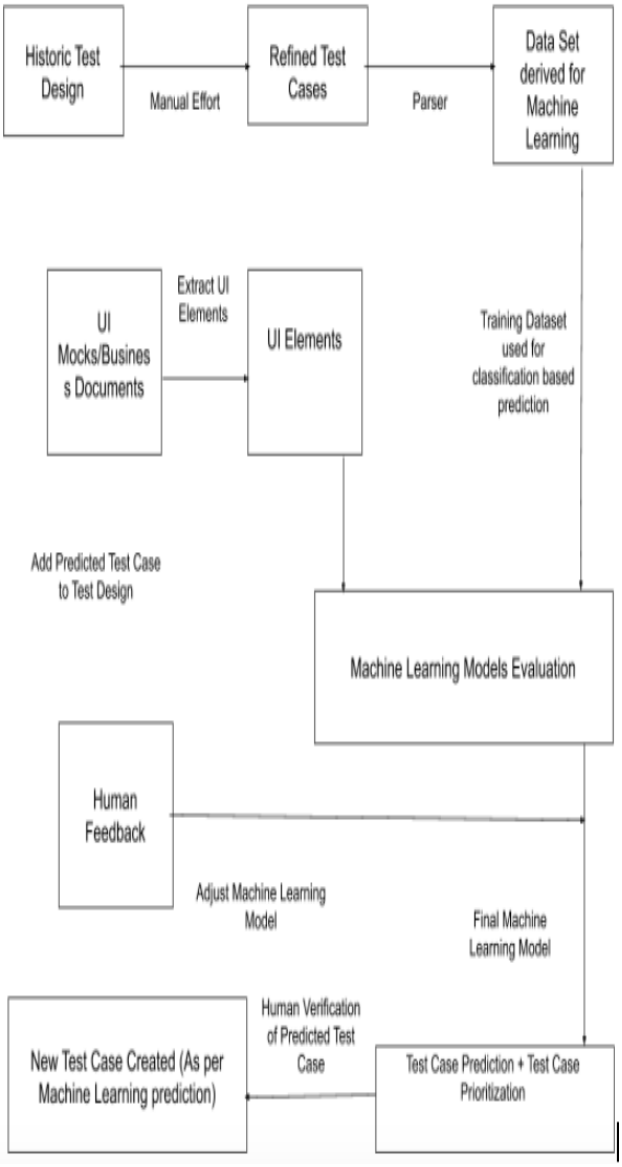

Fig.3 Complete flow diagram

Now after the training and getting the validation results with precision we used a new page data set i.e. Registration Page to test the prediction on input data. The results came out to again show that K-Nearest Neighbors Classifier algorithm was the most precise in predicting that the Contact page & Careers page were most similar to the Registration Page. Below are the results of few attempts to predict the same successfully

When the predicted results came we referred to the historical test cases and their priority for Contact and Careers page. This gave us an insight that how many test cases for high, medium and low priority needs to be created. For example: Contact & Careers page both has 2 High Priority tests. First test is to fill up the information on respective pages and the second one is to submit the information, which actually gets delivered via mail.

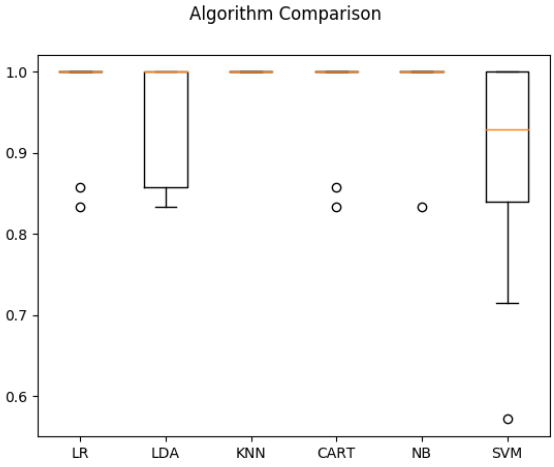

Fig. 4 Algorthim Comparison 1

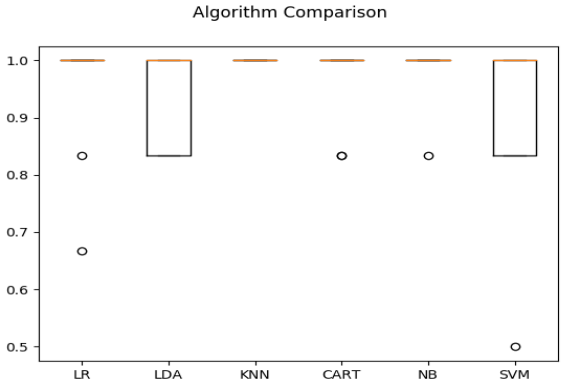

Fig. 5 Algorthim Comparision 2

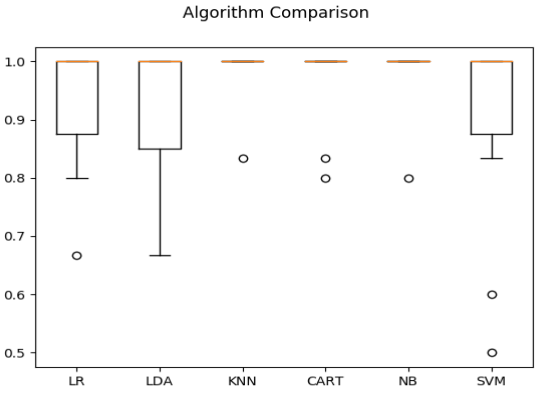

Fig .6 Algorthim Comparison 3

Here is one of the prediction results in tabular format:

Table-X Prediction Result

Page	Precision	Recall	F1-Score	Support
About	.94	0.8	.89	5
Careers	1.0	1	1	7
Cart	0.88	.92	.93	7
Contact	1.0	1	1	5

When the predicted results came we referred to the historical test cases and their priority for Contact and Careers page. This gave us an insight that how many test cases for high, medium and low priority needs to be created. For example: Contact & Careers page both has 2 High Priority tests. First test is to fill up the information on respective pages and the second one is to submit the information, which actually gets delivered via mail.

This gave us a starting point that the new Registration Page should have at least 2 High Priority test cases. Seeing that all these three pages largely serve the same purpose i.e. to fill some information and then submit it. This showed that our prediction was accurate and it logically was also valid. So we used this information to devise high priority test cases for Registration Page. This proves that the test case design [19] and prioritization can be successfully predicted using our method described above.

IV. CONCLUSION

The relation between historical test cases, user interface element types (present on the user interface pages for which these test cases were written) and the respective count of these user interface elements for each respective page, can help us predict how many test cases, and of which priority needs to be written for any given new user interface page (i.e. planned to be developed or is already under development currently).

Through experimentation we came to understand that how can we establish this relationship. We were able to learn that when on a particular user interface page if there are certain numbers of user interface elements. Then we can map the count of different user interface element types to the respective user interface page. Each of this user interface page is already covered in historical test cases. Now we know which user interface elements are present on which user interface pages and which all-historical test cases (these test cases already have their priority defined in the past) covers these pages.

This relation can help us build a classification based prediction model in which if we enter the number and type of user interface elements for a new user interface page. It will evaluate with the help of K-Nearest Neighbor classification and then provide us with an already existing user interface page to which this new page is most similar. We can now go back and find which all historic test cases were designed for the existing user interface page and thus we can use this information to create test cases and prioritize them for this new user interface page with very less manual effort. Thus significantly reducing man-hours and cost involved in developing this new user interface page, which is part of a software system or module.

In this paper we demonstrated that how we can use classification based machine learning [6] i.e. K-Nearest Neighbor classification to significantly speed up the test case design [19] and test case prioritization [8] using historical test case data [9]. This method will prove to be very helpful for software testing teams who wish to quickly write test cases for new software systems or modules.

This methodology or process can be improved further in future by utilizing the keywords and user interface element names used in the historical test cases to predict even more precise and possibly even create test cases by its own. Mapping user interface elements keyword occurrences in historical test cases with the priority of the historical test cases [14], and the number of different user interface elements present on the respective page would better predict which future page closely relates to the respective page and hence help in precise test case designing. We are working in this direction and also wish to integrate this with Software Development and Bug tracking systems to suggest possible test cases by analyzing the keywords in the respective task ticket and historical test cases for similar features or functionalities developed in the past. We believe that efforts in these directions will provide us with more fruitful results.

REFERENCES

1. R. Lachmann, S. Schulze, M. Nieke, C. Seidl and I. Schaefer, "System-Level Test Case Prioritization Using Machine Learning," 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), 2016, pp. 361-368. doi: 10.1109/ICMLA.2016.0065
2. H. Spieker, A. Gotlieb, D. Marijan, M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration", Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 12-22, 2017.
3. Y. Fazlalizadeh, A. Khalilian, M. A. Azgomi and S. Parsa, "Prioritizing test cases for resource constraint environments using historical test case performance data," 2009 2nd IEEE International Conference on Computer Science and Information Technology, 2009, pp. 190-195. doi: 10.1109/ICCSIT.2009.5234968
4. Md Junaid Arafeen and Hyunsook Do. Test case prioritization using requirements-based clustering. In ICST'13. 312–321.
5. Zhang, D., and Tsai, J. J. P. (2005). Machine Learning Applications in Software Engineering. River Edge, NJ: World Scientific.
6. Perini, A., Susi, A., Avesani, P.: A Machine Learning Approach to Software Requirements Prioritization. IEEE Transactions on Software Engineering (2012)
7. Martin Fowler and M Foemmel. 2006. Continuous integration. (2006). http: //martinfowler.com/articles/continuousIntegration.html
8. Benjamin Busjaeger and Tao Xie. 2016. Learning for Test Prioritization: An Industrial Case Study. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, New York, NY, USA, 975–980. https://doi.org/10.1145/2950290.2983954
9. J.-M. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In Proc. ICSE 2002, pages 119–129.
10. T. B. Noor and H. Hemmati. A similarity-based approach for test case prioritization using historical failure data. In Proc. ISSRE 2015, pages 58–68.
11. S Chen, Z Chen, Z Zhao, B Xu, and Y Feng. 2011. Using semi-supervised clustering to improve regression test selection techniques. In 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. IEEE, 1–10. https://doi.org/10.1109/ICST.2011.38
12. F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. 2011. Scikit-learn: Machine Learning in {P}ython. Journal of Machine Learning Research 12 (2011), 2825–2830.
13. H Park, H Ryu, and J Baik. 2008. Historical Value-Based Approach for Cost Cognizant Test Case Prioritization to Improve the Effectiveness of Regression Testing. In 2008 Second International Conference on Secure System Integration and Reliability Improvement. 39–46. https://doi.org/10.1109/SSIRI.2008.52
14. Khalilian, Alireza & Abdollahi Azgomi, Mohammad & Fazlalizadeh, Yalda. (2012). An improved method for test case prioritization by incorporating historical test case data. Science of Computer Programming. 78. 93–116. 10.1016/j.scico.2012.01.006.
15. S Stolberg. 2009. Enabling agile testing through continuous integration. In Agile Conference, 2009. AGILE'09. IEEE, 369–374.
16. V. H. S. Durelli *et al*., "Machine Learning Applied to Software Testing: A Systematic Mapping Study," in *IEEE Transactions on Reliability*. doi: 10.1109/TR.2019.2892517
17. J. C. Westland, "The cost of errors in software development: Evidence from industry," Journal of Systems and Software, vol. 62, no. 1, pp. 1–9, 2002.
18. M. Bowles, Machine Learning in Python: Essential Techniques for Predictive Analysis. Wiley, 2015.
19. V. H. S. Durelli *et al*., "Machine Learning Applied to Software Testing: A Systematic Mapping Study," in *IEEE Transactions on Reliability*. doi: 10.1109/TR.2019.2892517
20. L. C. Briand, Y. Labiche, Z. Bawar, and N. T. Spido, "Using machine learning to refine category partition test specifications and test suites," Information and Software Technology, vol. 51, no. 11, pp. 1551–1564, 2009.
21. M. Noorian, E. Bagheri, and W. Du, "Machine Learning based Software Testing: Towards a Classification Framework," in International Conference on Software Engineering and Knowledge Engineering (SEKE). Knowledge Systems Institute Graduate School, 2011, pp. 225–229.

AUTHORS PROFILE

Mayank Mohan Sharma, has more than 12 years of experience working in high tech research field (mainly in nanotechnology, embedded systems, mobile applications and software development). During these years he has also worked at technology startups and leading software technology companies. He is currently working as Principal Software Test at Zillow Inc., located in San Francisco, California, USA. He has an undergraduate engineering degree in computer science and engineering. He has won many awards for his outstanding contributions and got accolades from several renowned and known industry leaders. He is published author of two books; one on software testing in agile environments (which is best seller in agile methodology related books), and the other on crowd sourced testing management. He also has two patents pending in application of machine learning in test automation & test designing. His areas of interests are machine learning, artificial intelligence, software testing and software development.

Mr. Akshat Agrawal, working as an Assistant Professor in Amity School of Engineering & Technology, Amity University Haryana. He is having total 10 years of teaching & research experience. His major research interest in Deep Learning & Machine Learning, Artificial Neural Network, Speech processing and Image processing. He has been published 15 research papers in reputed international journals. He has been guided 12 MTech thesis and 30 BTech projects.

Dr. B. Suresh kumar,working as an Assistant Professor in Amity University, Rajasthan having Rich experience of 12+ years as an academician. Published 20+ research papers in Journals and Conference Proceedings of International & National repute. Member of International Association of Computer Science and Information Technology (IACSIT) *Singapore*; Computer Science Teachers Association (CSTA), Member of *ACM-USA*; Computer Society of India (CSI). Research interests include Nature Inspired Computing, Networks, Knowledge Management