1.4 Merits of Automated Testing

1) It is faster. After the initial time given to generate the test scripts, the execution of automated tests is much faster.
2) It is more reliable. Once test scripts are written and added to test suite they cannot be forgotten where tester can forget to perform some specific tests. Also automated tests are more accurate than manual tests as they do not involve human errors.
3) It reduces human and technical risks. If the developer team changes automated test scripts will help them reuse the previously developed tests and thus reduce the risks.
4) It is more powerful and flexible Using manual testing we cannot create 100 virtual users at a time which can be done by using automated testing. Also the test scripts can be reused.

2. Related Work

Conventional approaches for test automation include record-replay and keyword-driven automation, discussed in the Introduction. Record-replay, which is a feature available in many commercial and open-source tools (e.g., RFT [8] and Selenium [7]), requires a human to perform the manual test steps on the application user interface. Keyword-driven automation involves the creation of a library of reusable subroutines or keywords. A manual test case is translated (by a human) into a sequence of keywords; a driver program interprets such sequences by invoking appropriate Subroutines. Both these techniques require human interpretation of English language tests, whereas our approach attempts to eliminate this and can work in a more unattended manner.

There is a large body of work on synthesizing programs via mechanical interpretation of natural-language phrases, which is inspired by the ideal of bringing programming to end-users and bridging the gap between human-readable natural languages and mechanically interpretable programming languages [1][3]. Some of limitations arise from a test case being too specific.

The CoTester system [13] uses an English-like testscripting language, called ClearScript, which can be automatically interpreted. CoTester provides a record-replay feature similar to that available in testing tools, with the difference that the recorded scripts are in the stylized English form of ClearScript and, therefore, easily readable even by non-programmers.

Test cases writing in the form of tuples in web based application to give an ease to the non technical user. Each script step is a tuple consisting of an action, the target GUI element for the action, any associated value, and some metadata. [2]

Test merging technique for GUI tests. Given a test suite, the technique identifies the tests that can be merged and creates a merged test, which covers all the application states that are exercised individually by the tests, but with the redundant common steps executed only once.[4]

Test automation usually requires substantial upfront investments, automation is not always more cost-effective than manual testing. To support decision-makers in finding the optimal degree of test automation in a given project, we propose in this paper a simulation model using the System Dynamics (SD) modeling technique[5].In this paper study was to investigate how the simulation model can help decision-makers decide whether and to what degree the company should automate their test processes.

3. Proposed System and Framework

In this paper, we are trying to produce automated test script from manually given input in the form of keyword table. Also this can be realized using a sequence of keywords which will automatically call their function calls. These functions can be created by any non-technical user with the help of given UI and adding sequence of tuples formed by given keywords. These test cases created from UI in natural language are saved in DB.

Our task is to 1. Interpret the test cases written in natural language. 2. Create test scripts ready for execution. Parsing of Keywords, action and data is done. The test steps in keyword script will be dispatched to our test application framework to generate test scripts automatically for final execution. [1]

Table 2: A script consisting of key words and specified actions, their targets and any necessary data

Action	*Target*	*Test Data*
Open		http://localhost:8080/main.jsp
Select	Student Login	"PVPIT"
Enter	Username	"PVPIT"
Enter	Password	"ME2015"
Click	Login	
Exists	ME 2015 Batch	

Table II shows the data given by the user in UI application and this data is used by the generic automation framework from execution of test cases. This frame work gives user to add flexibility of adding and creating new keywords. It also facilitates execution of java code or shell scripts at specific step.

A. System Architecture

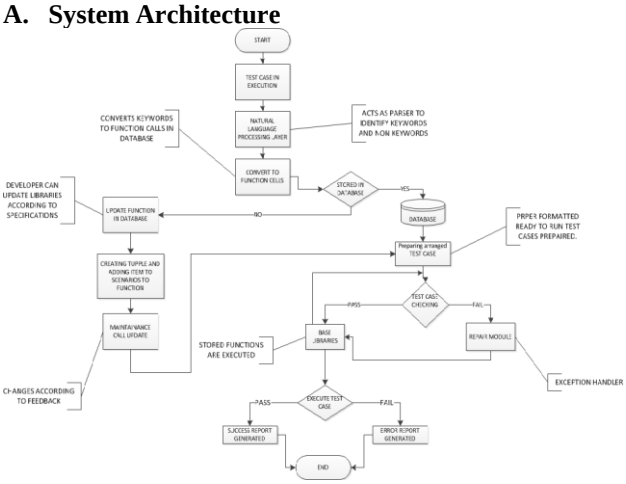


Figure 1: Proposed system architecture

Our system will try to give complete process from test case generation to test script formation and their execution under automation framework generating report for the test.

With this work we will bridge the gap between novice programmer and testing domain. Tester need not need to learn any specific programming language to automate test cases. He need just to fill the excel sheet in natural language which serves as keyword script to our framework system. After that generic test automation system will do the rest of work along with producing final report. There is a test data module which will directly enter the default data given in dataset for the given keyword. If data is not present in default set then system will give appropriate error and user has to enter specific data in framework data set. Otherwise, the tool will work without human intervention.

Reusability of test scripts by making test function of selected few steps will reduce maintenance cost of test cases. Fig. 1 shows the overview architecture of the system and the structural integration of the modules. Test cases in excel file is fed as input that is processed and saved in database in better manageable way and avoid redundant test cases steps. This is then executed under our frame work as per instructions in test steps. While execution if some test case fail due to locator changes then our frame work will try to resolve the problem and execute the test case. If recovery is not possible by our recovery module it passes the control to error handler here the error detail and logs are maintained which are used in reporting module.

B. Mathematical Model

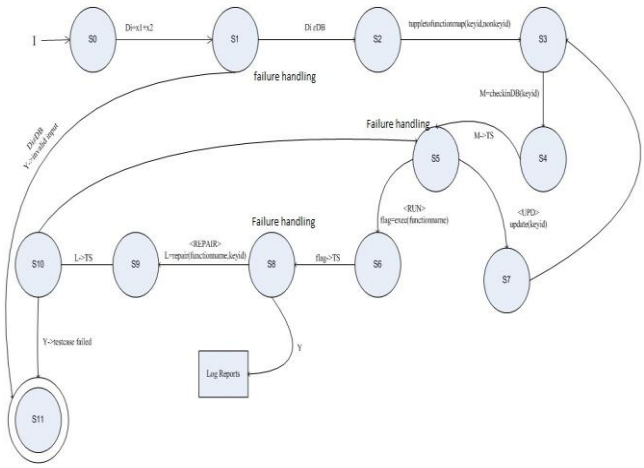


Figure 2: Mathematical model

Let S be the system consisting of solution of problem.
S={S, I, S11, D, DB, F, Y, TS}
S0=start state of software
Establish connection between software & internet
S11=end state
F=set of failure handling states
={ S1, S5, S8}
DB=Database
TS=test script generator
I=set of inputs
={X1, X2, X3}
X1= Set containing keywords
X2=set containing non-keywords
X3= button click to give input to software
Y=output of actions executed
X1, X2, X3 -> Y

Algorithm 1: The Algorithm for generating a test script from a test case

```
Input: Testcase tcase
Output: Testscript tscript or null
1.  while true do
2.      Invoke ExplorePath algorithm
3.      if successful then
4.          Collect generated test script tscript;
5.          return tscript
        /* Choose a new path in the test-flow graph */ ;
6.      Get the last decision point;
7.      if last decision point has more alternatives then
8.          Move to next alternative and goto Step 1;
9.      if not last decision point is root node then
10.         Set previous decision point as last decision point;
11.         Goto step 7;
        /* Explored all paths in the test-flow graph */ ;
12.     return null;
```

Algorithm 2: Explore Path

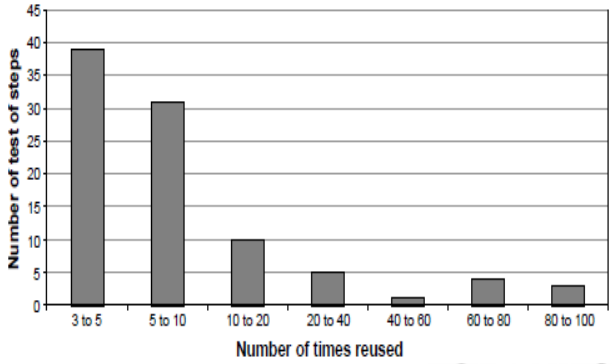
```
Input: Testcase tcase
Output: Success or Fail
1  foreach test step tstep ∈ tcase do
2      Compute segment lists for tstep (if not already computed);
3      Get current segment list of tstep;
4      foreach segment seg ∈ segment list do
5          Generate tagged segments for segment seg;
6          Get current tagged segment tseg of seg;
7          Generate tuple list tuplist for tseg;
8          if tuplist list is empty then
9              return FAIL
10         foreach tuple tle ∈ tuplist do
11             Identify UI element list for target t of tle;
12             if element list is not empty then
13                 Get current element uelem of tle;
14                 Execute action a of tle on uelem;
15                 if action a is not successful then
16                     return FAIL
17         if at least one tuple is successful then
18             declare segment seg as successful;
19     if no segment is successful then
20         return FAIL
21 return SUCCESS
```

4. Experimental Results

We analyzed 5547 steps over all scripts. Among these, NLP generates the desired sequence of tuples as the first choice for 1636 (29%) steps. For another eight steps, NLP generates the desired sequence, but not as the first choice. These results indicate that, in many cases, the first choice generated by NLP is the desired one, and users need not browse the other choices. Among the remaining steps, users entered feedback for 1886 (34%) steps, and ATA reused the feedback for 2017 (37%) steps.

Figure 5 presents the reuse data in more detail: it shows the number of test steps for which the amount of reuse falls in

different ranges. For example, the last bar in the chart illustrates that, for three steps, the feedback is reused between 80 and 100 times (in this case, the actual reuse numbers are 87, 88, and 92). Similarly, for more than 30 steps, reuse occurs 5–10 times. For another 192 steps (not shown in the figure), reuse occurs once or twice. Overall, the results demonstrate that similar test steps occur frequently and, therefore, that feedback-based reuse is a valuable feature.



To evaluate the effectiveness of ATA's script-repair capability, we executed the automated scripts of APP using different configurations, such as different browser version, different browser type, and different application version. In particular, we study the following research questions: (1) How often is ATA able to repair the scripts automatically? (2) What are the scenarios in which ATA fails to repair the scripts.

CHARACTERISTICS OF TEST SCRIPTS.

	$T_{(APP_1, V_1, IE_8)}$	$T_{(APP_2, V_1, IE_8)}$	Total
Manual tests	581	305	886
Automated scripts	392	148	540
Total script steps	6386	4285	10671
Average script length	16	29	-
Maximum script length	87	112	-

Figure 3: An illustration of the estimated ambient noise floor with an increasing number of senders.

5. Conclusion

We designed an error-minimizing-based framework to localize jammers. Most of the existing schemes for localizing jammers rely on the indirect measurements of network parameters affected by jammers, e.g., nodes hearing ranges, which makes it difficult to accurately localize jammers. In our method we localized jammers by exploiting directly the jamming signal strength (JSS).In particular, we combined the centroid based localization with the existing error minimizing framework. By combining these two methods we can achieve the better result to locate the jammer in wireless sensor network.

6. Future Work

In the future, test script can be generated on its own with just giving input in natural language statements. Reusing common steps in hybrid model of framework.

References

[1] "Automating Test Automation" Suresh Thumma lapenta, Saurabh Sinha, Nimit Singhania, and Satish Chandra† IBM Research – India (2012)
[2] "Efficient and Change-Resilient Test Automation: An Industrial Case Study" Suresh Thumma lapenta, Saurabh Sinha, Nimit Singhania, and Satish Chandra† IBM Research – India (2013)
[3] "Automatic Early Defects Detection in Use Case Documents" Shuang Liu, Jun Sun, Yang Liu, Yue Zhang, Bimlesh Wadhwa, Jin Song Dong and Xinyu Wang - ASE 2014.
[4] "Efficient and Flexible GUI Test Execution via Test Merging" Pranavadatta Devaki, Suresh Thummalapenta, Nimit Singhania, Saurabh Sinha IBM Research – India (2013).
[5] When to Automate Software Testing? Decision Support Based on System Dynamics: An Industrial Case Study" Zahra Sahaf, Vahid Garousi, Dietmar Pfahl, Rob Irving, Yasaman Amannejad - ICSSP 2014
[6] "Automated Testing of Industrial Automation Software: Practical Receipts and Lessons Learned" Rudolf Ramler, Werner Putschögl and Dietmar Winkler, MoSEMInA - 2014.

Author Profile

Vishal Sangave, received the B.E degree in Computer Science & Engineering from GSM College of Engineering, Pune in 2010 and is currently pursuing his M.E (CS) in PVPIT Bavdhan, Pune. His area of interest lies in Automation Testing and Artificial Intelligence.

Vaishali Nandedkar, working as Asst. Prof. at PVPIT Bavdhan. Hold a Master degree in Computer Science. Her interests in teaching and Cloud Computing.