International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-6, Issue-6)

Research Article | **June 2017**

GUI Test Script Repair in Regression Testing

G. Geetha Priya
PG Scholar, Dept. of Computer Science & Engineering,
Sree Vidyanikethan Engineering College (Autonomous),
Tirupati, Chittor District, A.p, India

Dr. B. Narendra Kumar Rao
Professor, Department of Computer Science & Engineering,
Sree Vidyanikethan Engineering College (Autonomous),
Tirupati, Chittor District, A.p, India

Abstract–

GUI testing is a process of testing a software application or functionality of GUI. It is defined as an interface between user and software which provides an easy way to interact with the system. GUI plays an important role in software engineering. GUI-based application requires that a test case consists of sequences of user actions/events to be executed. Selenium is an open-source tool for testing GUI Application by executing test cases whether the GUI Application is working properly or not. In this, we present test script repair technique to repair test cases. Test script repair technique uses reverse engineering process for creating the test script. Test script repair consists of three stages; they are Ripping, Mapping and Repairing. In ripping stage, there are two relationships for representing event interaction of GUI Application. During ripping we know the location of each widget. In mapping stage,original GUI events are mapped to an event-flow graph (EFG). In repairing stage, Event flow graph uses repairing transformations and human input to modified script to repair the test cases, and synthesizes a new "repaired" test script. During this process, test script repair uses GUI objects for yielding a final test script that can be executed using selenium tool to validate the GUI Application. An experiment using selenium tool to test, test cases suggests that it is effective in that unusable test scripts are repaired. Annotations significantly reduced the human cost to repair test cases.

Key Words- GUI Application, May-Follow relationship, Dominant relationship, Missing event, Missing edge, Test script Repair.

I. INTRODUCTION

GUI is an interface between user and software which provides an easy way to interact with the system through graphical symbols like icons and visual indicators and allows instead of a text-based user interface, text character and text navigation. It also consists of dialog boxes, icons, menus, scroll bars. GUI plays a very important role in software engineering which can interact with the user and program. GUI testing is defined by using different models like Waterfall model, Incremental model, V-model, Iterative model and Spiral model. Each model has its own life cycle to achieve the software development process.

Graphical User Interface (GUI) testing is a process of testing the functionality of GUI applications. It evaluates the designed elements such as layout, colors, fonts, font sizes, text formatting, captions, buttons, lists, icons, and links. GUI testing is defined in two types. One is manual testing and other is automated testing. GUI testing means checking the availability of controls (fields, buttons, check box, drop-down boxes etc.) in the application.

GUI testing requires a lot of programming which is either manual or automatic. There are two types of interfaces in computer application. One is command line interface which means where you type the text and computer responds to that particular command. Other is a graphical user interface which means where you interact with the computer using images. It will check and control the screens of menus, buttons, icons, bars, toolbar, menu bar, dialog boxes, Windows etc. A user can see only visible effects especially focused on design structure whether it is working properly or not.GUI testing is used to find out the applications interactivity and complexity.

GUI front end is that unacceptably large number of test cases will become unusable each time when the software is modified which is a major problem with test automation. GUI is unavoidable; due to the interaction with the software is the GUI. System testing is the process of testing the software as the whole which means even though the software is modified. GUI test case contains the sequence of user events or actions to be executed on software via widgets and check points and defines whether the software working correctly or not. These system tests are to be tightly coupled with the GUI structure that refers to the widgets in GUI and encodesa sequence of events like first click on file menu then click on new and open a selected file which is allowed in the GUIs workflow.

Due to the changes in the GUI, some of the test cases become unusable due to the event sequences which encode in the sequence are no longer works on the modified GUI and assertions which does not check correctly to the GUI objects.When the testing is done manually unusable test cases are not problematic because testing is done by human in order to execute the test cases according to the test plan and verify the correct output manually on the GUI that differs from the test plan and use common sense for the simple change in the GUI which encounters a bug to be revised for the test plan.

Unusable test cases are the main problem with the test automation due to the automated test harness which encounters unexpected widget different from what expects from the output it just hangs or execution fails. The process of

automation is desirable as the tested scripts can run a number of times which leads to a high cost for the maintenance when the GUI is modified a large number of tests become unusable which require re-recording and re-coding.

The only problem with the unusable test cases is thatthe main focus is on events represents as widgets, not as scripts and generated test cases automatically using model- based approaches when the software is modified and use a new model based test cases developed using repairing transformations which is matched to the unusable to usable model level test cases.GUI tests that are used in industry are coded as scripts and manually recorded and replayed using a test harnesses tools like quick test professional QTP, selenium,VB script which has a limited use in industry on manually scripted and captured test cases. But these scripts suffer from maintenance and need to be repaired using use cases and functional requirements. Testers invest large amount and time presenting their knowledge and experience in the test scripts which is complex and valuable as well.

Previous techniques cannot apply to the manually scripted test cases for a number of reasons. They are:

(1) The Event-Flow Graph EFG that forms the basis for the repair is inadequate as it is lacked some information like states, annotations that is required for the test script.
(2) Event-Flow Graph from the GUI using reverse engineering on depth-first traversal was too limiting. GUI ripping suffers from incompleteness which leads to partial EFGs. Previous work does not suffer from limitation because model-based test cases obtained from partial models where no possible events absent from models.
(3) Assuming the perfect knowledge of GUI changes which is not available to make perfect repairing transformations that are usable.
(4) Ignoring the test oracles that are assertions and checkpoints that playa vital role in the test scripts and to determine whether the test case passed or failed.
(5) Four transformations to define the changes in the GUI are limited and the tools that works are at the model level not at the script level.
(6) GUI test scripts are to be coded as low-level scripts like in QTP Window("PMS").Button("Add").Click to click onthe "Add" button in the window entitled "PMS". Mechanisms to automatically abstract scripts to model the transformations and synthesize low level to models.

A technique to repair the unusable test scripts to usable test scripts works by the level of the concept from script level to the model level and applies to the model based repairing transformations to be model level test cases and get a new low-level usable scripts and performs the repair with the EFG models of the GUIs. To repair test script takes an application as input A_0which automatically use reverse engineering EFG $G_0$of the original event and the modified version A_1 and the EFG G_1 and the test suite TS_0 which contains assertions and checkpoints createdon A_0. It also identifies $ts_0 \subseteq TS_0$ of test cases which are no longer usable for A_1 for repair and uses repairing transformations and human input to repair the test cases.

Test script repair which has three important points:

(1) Manually the test cases that are created often have more coverage than other applications that is automated using reverse engineering techniques.
(2) Repairs test cases successfully without breaking the original test cases and
(3) Human input as checkpoints in the model to repair over all repair process.

A. V- Model Testing Level:

1. Verification phase:

Verification phase can be defined in four phases. They are given below

a. Requirement analysis:

This is the first phase in V-Model or software development life cycle. It involves detailed communication with the user to understand his expectations and exact requirement. This is a very important activity for gathering exact software requirements. The acceptance test design is done at this phase and also used as an input for acceptance testing.

b. System Design:

Once we gather the clear and detailed product requirements, it will design the complete system. System design would comprise the details of complete hardware and communication setup for the product under software development with complete understanding and System test plan is developed based on the system design.

c. Architectural design:

Architectural requirements is understood and designed in this phase. Many approaches are proposed based on technical and financial feasibility. System design is broken down into modules based on different functionality. The data transfer and communication between modules with other systems is clearly understood and defined in this stage. The Pictorial representation of the design can be defined in this phase. This is also called high-level design (HLD).

d. Module design:

Detailed internal design for all system modules is specified in this phase, it is important that the design compatible with other modules in the system architecture and other systems. Unit testing is a very important part in the development process and it helps to eliminate maximum faults and errors at an early stage.

Coding Phase:

System modules designed in the design phase, in this suitable programming language is decided based on the system application and architectural requirement. Coding is performed by using coding guidelines and standards. Coding phase combines with both verification and validation phases.

2. Validation phase:

Validation phase can also define in four phases. They are given below

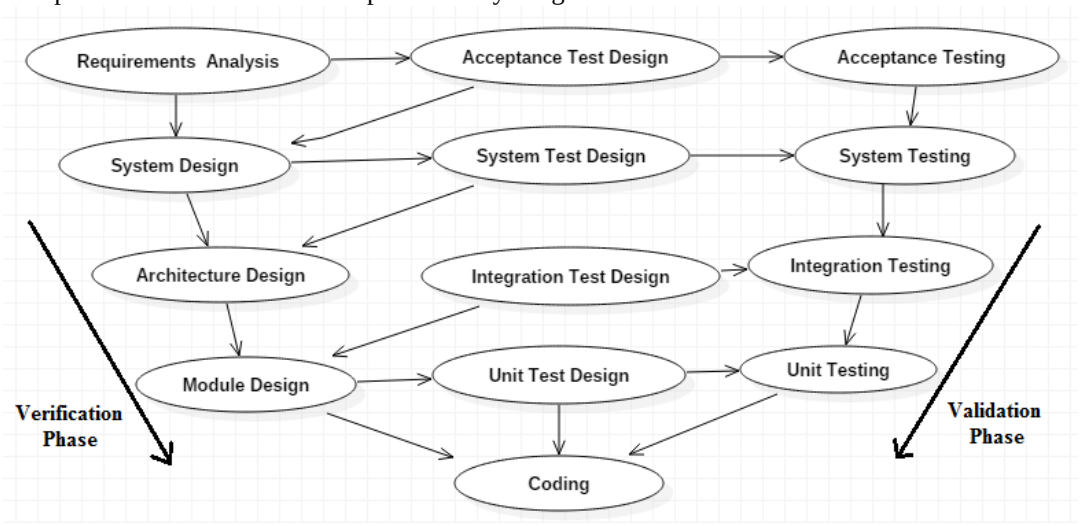

Fig 1: V- Model Diagram

a. Unit testing:

It is designed in module design phase and also executes the code during this validation phase. Unit testing helps to eliminate bugs at an initial stage and all defects cannot be revealed by unit testing.

b. Integration testing:

It is associated with the architectural design phase. Integration testing is to test the existence-communication of internal modules within the system application.

c. System testing:

System testing is directly associated with the System design phase. It can check the entire system functionality and communication under development of external systems. Most of the software and hardware compatibility issues can be revealed during system test execution.

d. Acceptance testing:

Acceptance testing is associated with Requirement Analysis phase. In this phase, a product is tested in the user environment and reveals the compatibility issues with other systems. It also identifies the non-functional issues such as load and performance defects in the user environment.

B. Approaches Of GUI Testing:

In GUI testing, a technique can be used to find out the errors in a particular product by using different approaches. GUI testing is used to test the functionality of GUI either manually or automatically. In GUI testing, various approaches are proposed and have their own advantages and disadvantages.

GUI Testing approaches are defined in two types:

1. Manual GUI testing:

In this approach, graphical screens are checked either manually based on requirements of the user. Manual GUI testing is used to find out the faults in software. After comparing user goals, knowledge and points feedback will be taken from the user. It is also used to find out difficult bugs, which are not found in automatic testing and it is good for initial testing. It can take more effort for providing weak coverage.

2. Automated GUI testing:

Automated GUI testing checks the task automatically by comparing the same result with an expected output. It is the solution for all raised issues in manual GUI testing. This approach can be defined in three ways. They are a Capture-replay method, Unit testing and Model-based testing.

3. Capture-Replay:

This is defined in two ways. In capture, test data will be captured by using the automation tool. During playback, captured data is executed on Application under Test (AUT).

4. Unit testing:

Unit testing frameworks are JUnit and NUnit. This provides flexibility and supports the test driven development and also automatic test execution.

5. Model-based testing:

Model-based testing is used for generating test cases. The main advantage is to determine undesirable states from GUI by comparing two methods. A model means graphical format of the system application, which helps us to understand the system application behavior and generated test cases by using system requirements. It can test the user selected sections from GUI and execute the system applications using automatic or semi-automatic test case generation.

Charts and decision tables are the modeling techniques which are derived from test cases. Charts check the state of the system after giving some input. Decision tables are used to determine results after input is applied.

C. GUI Testing Tools:

The Following are open source testing tools available to conduct GUI Testing. These testing tools are used to generate test cases and test the test cases. In this, we use selenium tool to test the test cases of GUI Application.

Table 1: GUI testing tools

Name	Licensed under
Ascentia Itest	Proprietary
AutoIt	Freeware
Oracle application	Proprietary
Maveryx	GNU,GPL
QF test	Proprietary
Selenium	Apache
Silk Test	Apache
Test Studio	Proprietary
Watir	BSD
Auto Hot Key	GPL
Sikuli	MIT
Robot Framework	Apache
Dojo	BSD

II. LITERATURE REVIEW

A. Requirement & Analysis phase:

Memon et al. [10] proposed the result of a two-phase empirical study in which the evaluation and the comparison of applicability in automated component-based and visual GUI testing by using GUITAR tool and referred as VGT GUITAR which executes faster than VGT GUITAR. First, GUI mutation operators have created 18 faulty versions of an application in which both tools are applied in the experiment. Results from 456 test case executions in each tool shown, with statistical significance, the Component-based approach reports more false negatives than VGT for acceptance tests but VGT approach reports more false positives for system tests. Second, a case study is executed with larger open source applications, ranging from 8,803-55,006 lines of code. Results show that GUITAR is applicable in practice but has some challenges that are related to GUI component states. The results show that VGT GUITAR is currently not applicable in practice.

Based on the study results presented areas of future work for both test approaches and concluded the various benefits and drawbacks. The component-based approach is important and executes tests faster than the VGT approach. The VGT approach can perform visual assertions and it is perceived more flexible than the component-based approach. The combination of the two approaches is the most suitable in practice and hence warrants future research. Automated testing is used for Conceptualization and Evaluation of Component-based Testing Unified with Visual GUI Testing.

Robert et al. [20] presented a multiple-case study with the goal to investigate the state-of-practice of the GUI-based system and acceptance testing at six software development companies of changing context. GUI based system testing is well-known such that automated GUI based system testing and acceptance testing exists only on a small scale. The study of these testing has some problems like a limitation of the test tool, test costs will be high and customer involvement in testing. System and Acceptance testing mainly focus on validation of systems conform to its requirements on a high level of system abstraction and executing end user scenarios using GUI. No empirical studies can evaluate GUI based system and acceptance testing, which is performed in industry. Capture and replay tools like an open source web browser automation tool; selenium IDE is used to interact with GUI by identifying the GUI component level under the bitmap GUI that is shown to user and record mouse co-ordinates. Test scripts can break easily if the GUI layout or components changes. In order to avoid the problem with capture and replay tools test cases, automated should require high maintenance effort.

B. Architecture design phase:

Rita et al. [2] proposed a conceptual framework for comparing different testing approaches. It is a fully automated GUI testing technique in a systematic way. Fully Automated GUI testing approach comes under model-based testing and also provides a conceptual framework by comparing various testing approaches and testing the adequacy of GUI models. This can be performed on real Android Application. The Android ripping tool is used for comparing fully Automated GUI testing Technique in a systematic way.

Eman et al. [3] proposed model-based testing is derived from design models to develop GUI in the Model-Driven engineering paradigm. This approach is used for designing multi-platform user interfaces that are suited for small screen devices. DSM is used for generating or deriving some test cases and can be defined by using state charts; each state represents a presentation unit. A model-based testing technique tests GUI designed by using model-driven engineering. This technique is defined under System under test (SUT) through generating the test cases. It can be defined by using MDE (Model-Driven Engineering) approach. MDE testing is a multi-platform testing technique. It presents transition

based testing criteria for MDE environment. MDE is used to find out a proper model that can serve as the testing model. No tool is used for Model-Driven Engineering (MDE) transaction based GUI testing technique.

Gennaro Imparato [6] proposed GUI ripping technique with the combination of Input perturbation testing for improving the quality of Android Application. Now a day's mobile applications are becoming an integral part of our daily lives. For this application security and reliability are necessary for growing rich mobile apps and used to test mobile applications using GUI testing. It creates a model by using Application under Test (AUT). GUI testing approach is used for two tools to support android applications, one is Slum Droid and another is GUI analyzer. GUI analyzer is used to perform Input perturbation. Fuzz testing tool is used for generating Sequence of events to the applications on android Virtual Devices (AVD).

Dervish et al. [15] proposed a technique that is GUI Invariant Discovery and validation. In this, automated GUI approach is used to detect a set of state-based GUI Invariants to enhance both GUI specification and test suites. Model-based testing is used to test automated discovery and validation of state-based GUI invariants. GUITAR Replayer tool is used to determine infeasibility of test cases and decides failure point and each test case of failed event. The framework is capable of capturing important aspects of GUI by reducing the number of infeasible test cases.

Tomi et al. [22] proposed dynamic reverse engineering for automatically generating GUI models that is suitable for MBT. GUI component classification suitable for GUI automation and introduces some examples of efficient modeling for GUI applications in automation strategy. In this, comparison of various approaches for automated GUI models which suit for Model-Based testing. Model-Based Testing (MBT) manually design test models and need a huge amount of effort and knowledge in formal modeling. MBT reduces the maintenance cost compared to CR tools, an abstract test model creates manually based on requirements of the SUT and MBT tool used to generate and execute tests based on models and verify whether the SUT executes the requirements correctly or not.

C. System design phase:

Chia-Hui Lin et al. [4] proposed PATS: Parallel GUI Testing Framework for Android Applications. Developers used to spend a large amount of testing time for executing test cases. GUI ripper checks the state equivalence and depth of threshold. GUI testing is based on Master-Slave model. PATS analyzed under the cooperation of the master and slaves. Test cases are generated at run time by using Application under Test (AUT). In PATS testing efficiency can be improved at the same time, testing time also improved effectively with an 18.87 - 35.78% performance. In this GUITAR tool is used to support and dispatch the test cases to Slave nodes for AUT testing. GUI testing plays a vital role in verifying operation of GUI components.

Myra et al. [9] proposed GUI interaction testing; it is a new family of coverage criteria for grounded of combinatorial interaction testing. GUI interaction testing is determined by using model-based testing which comes under automated testing and test cases are automatically generated and systematically explore the impact of context. The key motivation of combinatorial techniques enables us to incorporate "context" into new criteria in terms of event combinations and sequence length. It can detect many bugs which are not detected earlier. GUI Ripper tool is used to analyze the GUI application and traverses the application's GUI hierarchically.

Lin et al.[12] used Markov chain based on statistical testing which play a significant role in the economic production of high-quality software which provides a fact in order to support dependability. The Sequence-Based specification gives a clear-cut condition that can draw from formal system model to informal functional requirements. This can be used as a formal method for constructing a chain usage model for statistical testing and also expresses a case study where two methods can apply in combination with supporting tools for fully automated statistical testing in GUI application. An initial result shows potential application method that makes possible with the tools which are developed in the software quality research laboratory. A prototype sequence enumeration tool is used for requirements elicitation and analysis with Sequence-Based Specification.

Chen et al. [13] used AD (Activity Diagram) based on Automated GUI testing framework which can support the user modeling behaviour, GUI test case generation, post-test analysis and debugging. An experiment shows that this approach not only reduces overall time nevertheless improve the quality of designs. Due to increase in complexity and market pressures, functions validation became a major block for Smartphone applications like android IOS. The execution of Smartphone applications intensely relies on interactions with the users. Manual testing is very slow and expensive. Due to the lack of formal models the user behaviours in the design phase delays in test case generation and test evaluation whereas detailed test efforts are required to make sure of the consistency between user specification and GUI implementations. The existing testing approach can utilize design phase for testing complex applications. In this, two industrial smart phone applications demonstrate that our approach not only reduces overall testing time, but also it can improve the quality of designs.

D. Module design phase:

Ke Shen et al. [5] proposed Automatic GUI testing Approach for Android Applications. This approach builds and consists of a dynamic GUI model of applications at run time, which is based on extensive non-deterministic considered transition system that shows the weight of transitions between GUI states. Extracting part of the GUI features of the application under test (AUT), the model consists of simple enough to avoid state explosion and improves the test efficiency, but provides directed guidance for testing event generation at the same time. A practical probabilistic search based event selection algorithm is used to leverage data provided by the model, transform weight of transitions to select the testing event to execute. The algorithm solves the non-deterministic issues introduced by the estimation of the model.

Empirical assessment of several real world applications can achieve high code coverage quickly and detect bugs efficiently. Android platform provides a fuzz testing tool called monkey runner, which can send random GUI events to the System under Test. python programming interface is used for executing GUI events. This approach is defined under automated Black Box GUI testing. Symbolic Execution approach is used often requires access to the source code that are not applicable for many proprietary Applications. It is tested by using a model-based GUI testing.

Kevin et al. [7] used Dynamic Symbolic Execution to generate inputs in Search based GUI testing. In this, a hybrid approach is used to describe search based testing for generating a sequence of events in GUI. DSE (Dynamic Symbolic Execution) is used to build the input data for text boxes. It can be achieved through generating test case using SUT (System under Test) for executing GUI. The Symbolical software application DSE can increase code coverage on two test case generation applications. EXSYST tool is used in GUI testing.

Shan et al.[11] proposed an approach which can automate the testing of android applications based upon capture and replay method. In recent years number of android applications has increased in order to improve the quality of the android applications testing. The user events of android applications can capture the events and converts into Robotium test script, which can replay recorded user actions. It also allows assertions when capturing and verifying the outputs of android UI components. For automatic functional testing of android applications, several toolshave been proposed like monkey runner, Robotium, UI Automate which testers manually create test scripts and execute tests automatically.

Harvinder et al. [14] proposed an approach to identify the anti-patterns in object-oriented systems during automation testing. These anti-patterns are considered as incomplete programming practices which are unacceptable as a solution. In software development certain patterns are unfavorable when compared to design patterns gives acceptable solutions to same problems. The impact in presence of anti-patterns examines the study on anti-pattern classes. Certain issues in testing strategies used for finding bugs in software systems are goals of testing, a procedure for testing new functions, greatest resource and time for execution of the project. There is a huge requirement to carry out testing process in an appropriate way due to difficulty in software development systems. Anti-pattern testing is used for reducing the testing cost in different modules. Argo UML is used on a path of GUI application and XMI.

Gagandeep et al.,[24]gives an analysis of the modeled based regression testing for web applications. Model based testing is the automation testing approach which generates more flexible and explicit the application. Web components play an important role in the evolution of rapid growth of services and the information through the internet. Websites and web services are a basic for the quality of service to provide the different functionality of their correctness. Test automation is the need for diversity and complexity of web system for the use of automatic execution in order to compare the expected output. Models describe the changes that are incorporated before and after the execution and helps to identify the test cases that are generated for the initial version and to be rerun after the modification of the component. All-paths coverage criteria is the regression test suite that is generated for the component. This approach gives the significant results within the optimized generation of test data.

*Datchayani M et al.,[*25] Graphical User Interface is persistent to the coverage that mostly half of the code in the software systems is to produce the required GUIs. In order to generate the test cases GUI based software systems is more complex and is required to include all the possible event sequence by the user. A single change in the GUI may lead to a large number of test cases become unusable which is the main problem of software systems. The solution is to make the analysis of original GUI events and identifies the unusable test cases to be reused within the modified GUI. Test cases that are reusable which are subjected to the repairing transformations and make the unusable test cases reusable and which helps to construct the event flow graph (EFG) by using depth first manner and make the unusable test cases usable within the event sequence.

Memon et al., [26] GUIs plays a major role of the software that is being developed and created by using rapid prototype where there is no reason for the effectiveness of the regression testing in GUI which is different from traditional software. Whenever the GUI changes or modified test cases of the original GUI is either unusable or reusable on the modified GUI. Test case generation is a very expensive process where the main goal is to make the unusable test cases usable along with the test suite event sequences. This process does not explore earlier to make unusable test cases usable where large number of test cases become usable for the GUIs and also presents a new regression testing technique which automatically makes unusable after modification of GUI Application and determines unusable test cases which are repaired and executed. Repairing transformations is used in order to repair the test cases in the final stage. The regression testing has four repairing transformations that have been implemented. The case study has four open source applications which explain that this approach is effective in the sense large number of test cases can be repaired along with time and performance, certain types of test cases are becoming unusable, and certain types of dominator events become unusable test cases when GUI is modified.

Zhongxing Yu et al., [27] proposed two approaches to improve the test suite based repair and perform more studies on the feasibility and effectiveness of test case generation in the over fitting issue enables the test suite repair on 224 bugs repository.Test suite repair is one of the widest families of techniques among different types of program repair techniques. Test suites are the input and output specifications that usually lack complete specification of the expected behavior of the program under repair. Patches are generated by the test suite based program repair techniques that pass the test suite however may be incorrect. Overfitting patches failed to generalize other test cases specific to used test suite. Results which identifies that test case generation change the resulting patch but not effective at improper patches and identifies the problems which are related to ineffectiveness finds the next research in order to build the test case generation techniques to repair the automatic systems.

Pakinam et al., [28] explores many strategies that come into view on test case generation and test data by using different models of model-based testing. Software testing mainly depends on three stages: (1) Test case generation (2) Test execution and (3) Test evaluation. Test case generation is the core of any type of testing process such that generated test cases requires the test data that is to be executed in which test data generation is not less than important to the test case generation. During the past decade automation plays an important role which reduces the overall time and effort that is spent for the testing process. UML models are the best share among those models. In order to optimize and reduce the test cases, many researchers have paid attention rather than generating them.

Indumathi et al.,[29]mainly focuses on set of algorithms that derives the dependency structures among functions and finding the exact number of dependents of each function in a system which gives high prioritize to test cases in an automated way that depends on using graph coverage values.Test case prioritization is the process of conducting the order of test cases eventually and aid to meet with two important aspects in software testing namely time and budget which enhances the fault detection rate. Due to improved fault detection rate results have found in earlier stage the results help us to find the problem earlier and deliver the stage within short period of time. Some of the techniques were used in past specifically Random order, greedy technique, fine grained and coarse grained but these techniques are considered the test case into test suite in an independent way. Dependency is the interaction and relationships between system and sub-system. Some of the techniques are done manually which derives the dependency structures among the test cases. By executing the more complex test cases in high priority fault detection rate will improve to establish the truth Siemens test suite is used.

Bestoun S et al., [30] proposed a new approach for GUI functional testing using Simplified Swarm Optimization (SSO) which is used to generate an optimized test suite by using Event Interaction Graph (EIG). Graphical User Interface (GUI) is the main part of programs which helps the user to interact with different computing devices like computers, mobiles, and to very small devices like watches. This interaction uses different tools and objects like images, text, label, menus etc. Software meets quality that is required by the user which is a crucial component that is to be tested with different types of GUI. Functional testing does not deal with the coding internals which is a proper interaction between the user and application. This proposed generation algorithm is based on SSO has proved its efficiency by comparing with other algorithms. This approach also manages and repairs the test suites b deleting the unusable event sequences which is not relevant. This approach is applied to the standard case study and proves its applicability.

III. MODELS AND ALGORITHMS

GUI testing is a process of testing a software application or functionality of GUI. It is defined as an interface between user and software which provides an easy way to interact with the system. GUI is the most important role in software engineering. GUI-based application requires that test cases, consisting sequences of user actions/events to be executed. Selenium is an open-source tool for testing GUI Application, by executing test cases, whether the GUI Application is working properly or not. In this, we present test script repair technique to repair test cases. Test script repair technique uses reverse engineering process for creating test script. GUI Test script repair consists of three stages, those are Ripping, Mapping, Repairing. In ripping stage, there are two relationships for representing event interaction of GUI Application. During ripping we know the location of each widget. In mapping stage,original GUI events are mapped to an event-flow graph (EFG). In repairing stage, uses repairing transformations and human input to modified text to repair the test cases, and synthesizes a new "repaired" test script. During this process, test script repair uses GUI objects for yielding a final test script that can be executed using selenium tool to validate the GUI Application.

GUI Test Script repair deals with three levels of abstraction:

(1) Execution space (2) Script space and (3) Model space

Execution space is where the application is executed, Script space is the testing script and model space is the generation of Event Flow Graph.

Ripping raises the level of execution to model.

Mapping raises the level from script to model.

Repairing raises the level from model to script

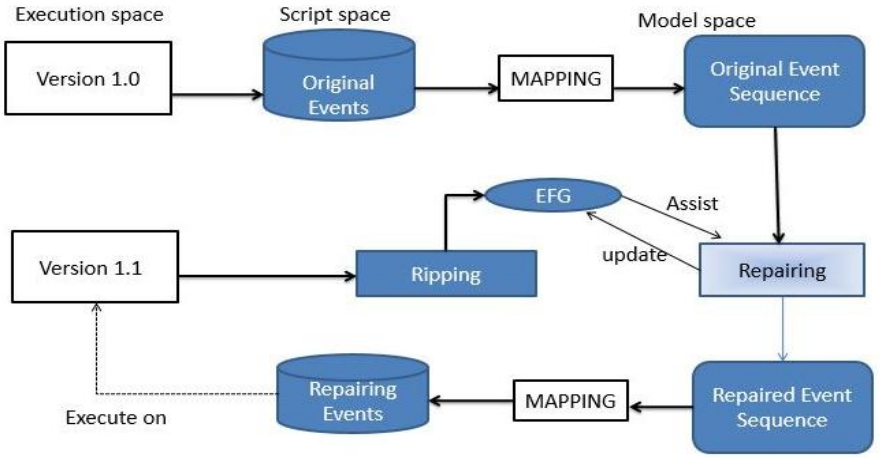

Fig. 2: Logical view of test script repair components

The three stages of GUI test script repair are

1. RIPPING:

GUI test script repair in regression testing is the main purpose is to repair test cases of GUI Application. First, we developed a GUI Application using the set of events. After creation of GUI Application, we need to generate test cases for that Application. Test cases are generated by using two techniques those are ripping and mapping. In ripping technique we define all events in GUI application. Those all events are defined by using may follow and dominate relationships. These relationships are helpful to find all events in the GUI application. May follows relationship will be performed in between two events. A May - follow edge relationship from vertex x to vertex y shows that an event y may be performed immediately after event x. Dominates edge relationship consist of a single incoming edge from vertex x to vertex y and vertex y must be preceded by after execution of vertex x. Ripping outputs an EFG model that represents GUI with all possible interactions. Modelled as a directed graph where vertices represent events and edges represent either may-follow or dominates relationship.

Ripping stage outputs an event flow graph for the classification of GUI events which is used to identify modal and modeless windows. Restricted-focus events are called modal windows and unrestricted-focus events are called modeless windows. If v is a vertex of restricted focus events open GUI windows that have a special property that once invoked, they monopolize the GUI interaction and if v is unrestricted focus events open GUI windows that do not restrict the user focus. Termination events close modal windows. If v is a vertex of termination event then it invokes the top-level events.

After ripping stage all EFG edges are marked as may-follow and to enhance the EFG a new type of edge called dominates edge which means that vertex e1 to vertex e2 shows that event e2 preceded by e1 has a single incoming edge from e1. In the process of test script repair in which some of the edges may manually update by the tester. Ripping stage knows the location of each widget like dialog, menus, checkboxes and class that used to create the GUI. These attributes address each widget where GUI events are searched in mapping from the ripper stage.

2. MAPPING:

Our mapping is a two-column lookup table: the first column is the addressing path that is used in EFG for an event; the second column is the path that we assign to the events in order to create the mapping. Firstly start with an empty mapping, adding entries to it as we iteratively examine each script using the following algorithm, which takes two inputs: (1) a test script TS consists of a sequence of script statements $<s_1,s_2,\ldots s_n>$ are created on the original version of the GUI Application and (2) the EFG of the modified Application. The output is the modified Mapping is defined below

1. If s_i already in Map, then skip to next script statement; else
2. If event belongs to graph such that event's GUI properties match with GUI properties from the GUI element in statement s_i, then add a new entry (s_i, e) to Map; else
3. add a new entry (s_i , NULL) to Map.

3. REPAIRING:

Each statement is mapped to GUI events in EFG model or NULL after mapping. Simultaneously GUI widget referenced in checkpoints in scripts mapped to model level or NULL. Now GUI test script is ready to start repairing.

The best case for the repairer is one in which test script contains non-NULL entry in the sequence such that maps for each of its events and there is an edge in the EFG for each of adjacent event pair. Such a sequence is considered to be valid. The mapped script must repair at least one NULL event in the sequence. Additionally, if there is no NULL event in script an invalid flow of events still wants to repair. The sequence of events is performed to the script is to be allowed by the GUI workflow.

Given an EFG G_1 and a mapped script which represents sequence of events (e1,e2...e12) and checkpoint (c1,c2...c12) where $e_i \in G1 U\{NULL\}$ *and* $c_i \in G1\ U\ \{NULL\}$, the script needs repair if one of the following two conditions is satisfied:

Repairing starts with four inputs:

(1) Set of event sequences in which each corresponds to test script,
(2) EFG in which all edges marked as may-follow
(3) Original mapping
(4) An empty table of approved paths

Test script repair can follows below two cases:

Case 1: If the sequence having missing event which represents vertex at least one of event $e_i$in the sequence is NULL;

Some of the statements in the mapping stage could not map to event sequence in the EFG. The reasons are:

1. The events may no longer exist in the GUI of assertion A_1- Tester needs to delete the event from the sequence and may-follow relationship will be added to an event prior to the current event.
2. The event may still exist in the assertion A1 but missed during automatic creation of graph G_1- Tester needs to add the missing event where the new event will add to the vertex set V and pair of new edges will be added to the may-follow edge set E_{may}.
3. Some of the attributes of the widget may have changed like new label may have been moved to the different location in GUI- Tester can re-map the missed event to the counterpartA_1 and the repaired information is added to the mapping table.

Case 2: If the sequence of events having at least one pair (e_i,e_{i+1}) of adjacent events in the sequence is not a valid edge in EFG.

Mechanisms to repair the event sequences:

1. Examines the approved path of two consecutive events x and y is found and searches in the look-up table and declares the two events are in the sequence and replaced with the subsequence.
2. Suppose if there is a dominates edge (a,y) in the EFG such that a is inserted into sequence before y and if no EFG edge (x,a) then this case applied recursively.
3. Repairs employs shortest path algorithm in order to find the paths between x and y- the paths from approved table is to be found by automatically tracing out dominates edge for repair.

This algorithm finds only the paths which have may-follow edges then confirmation is sought to tester:

1. Select one of the suggested paths.
2. Tester creates a new path by adding events and edges.
3. Join the new direct edge (x, y).
4. Declare y is not reachable to x then test scripts cannot be repaired.

Algorithm for Test Script Repair

1: Initialization of global variables:

G_1= (V, I, E), mappingTable= ∅, approvedTable= ∅

2: **Output:** TS_1, updated G_1, *mappingTable, approvedTable*

3: Procedure*repair* (TestScript TS_0):

4: $TS_1 \leftarrow \emptyset$

5: For all test cases *TC= <s_1, s_2, s_3, ….. s_n>∈ TS_0*

6: *Evtset* ← ∅; *Cptset* ←∅

7: For all statements or checkpoints s_i∈ TC

8: Map s_i to event e_i or checkpoint c_i
//mapping table

9: *Evtset.add (e_i); CptSet.add (c_i)*

10: **For all** events e_i and checkpoints c_i∈*Evtset* U *Cptset*

11: **If***e_i=NULL* or *c_i=NULL*

12: *repairedSeg* ← *repairEventAndUpdateModel (s_i, e_{i-1}, e_{i+1})*

13: *(EvtSet*U*CptSet).replace(<e_{i-1},e_i,e_{i+1}>, repairedSeg)*

14: **For all**<e_i, e_{i+1}>where e_i and e_{i+1}∈*EvtSet*

15: **If** <e_i, e_{i+1}>∉ E

16: *repairedSeg* ← *repairEdgeAndUpdateModel (e_i, e_{i+1})*

17: *(EvtSet U CptSet), replace (<e_i, e_{i+1}>,repairedSeg)*

18: *TS_1.add ((EvtSet*U*CptSet).mapToTestScript())*

19: **Procedure** *repairEventAndUpdateModel (s_i, e_{i-1}, e_{i+1})*

20: **If** confirm script s_i or remap to e_i*':*

21: add $s_i$or e_i*'* to *V* as e_i

22: add <e_{i-1}, e_i> and <e_i ,e_{i+1}> to E_{may}

23: **Return**<e_{i-1}, e_i ,e_{i+1}>

24: **If** delete Node:

25: add <e_{i-1}, e_{i+1}> *to* E_{may} if not exist

26: **Return**<e_{i-1}, e_{i+1}>

27: **Return** *repairEdgeAndUpdateModel (e_{i-1}, e_{i+1})*

28: Return< e_{i-1}, e_{i+1}>

IV. CASE STUDY

The overview of GUI test script repair presents a running example of MS-Word. It consists of 12 events that consist of menus which allow the user to select an option out of the list of options or commands. Microsoft word pad consists of menus like file, edit and help that selects the list in the menu.

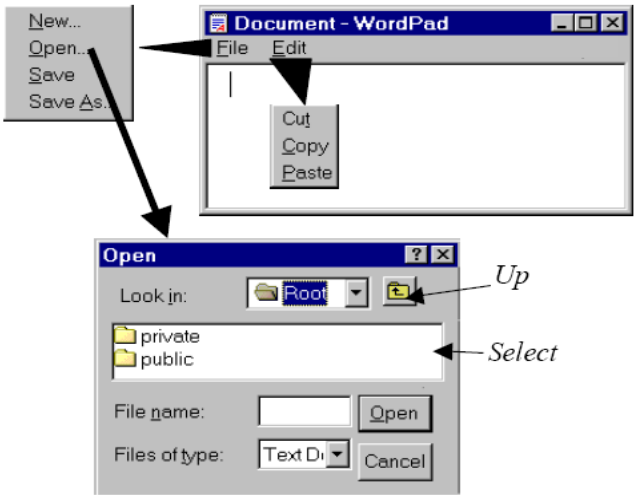

Fig.3: GUI example

GUI events= {File, Edit, Help
Open, Save
Cut, Copy, Paste
About, Contents
Ok, Cancel}

STAGE 1: RIPPING

We start by running our Ripperon the application in its start state to obtain its EFG. Events e1,e2...e12 are all available in the main window; their states, each represented as a set of triples of widget, property and value are shown in Fig. Because of theiravailability in the GUI's start state, these events form the initial nodes setI.

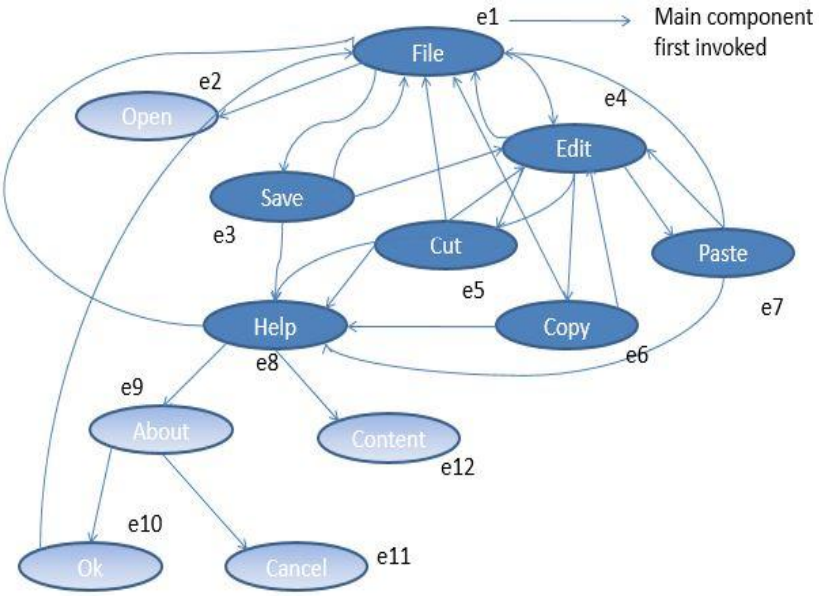

Fig.4: Ripped EFG

The Ripper incorporates these nodes into theEFG; The Ripperthen starts executing the encountered events one by one: e1 followed by e2, then e3, and e4. After events e1, e2 and e3 the Ripper determines that they are non-structural eventsbecause no window is opened or closed; the follows relationships are then computed according to the algorithmsand add to the EFG. Evente4 opens a new window; because of the selected state of the edit button and checked state of e1, the newwindow is titled edit with three events e5, e6, and e7, allenabled. They are all executed but no new window opens. Their follows relationships are then computed and added tothe EFG. The below figure shows final EFG after the Ripping phase.

Ripping outputs an EFG model that represents GUI with all possible interactions. Modelled as a directed graph where vertices represent events and edges represents either may-follow or dominates relationship.

A. May-follow relationship:

May-follows relationship will be performed in between two events. A May - follow edge relationship from vertex x to vertex y shows that an event y may be performed immediately after event x.

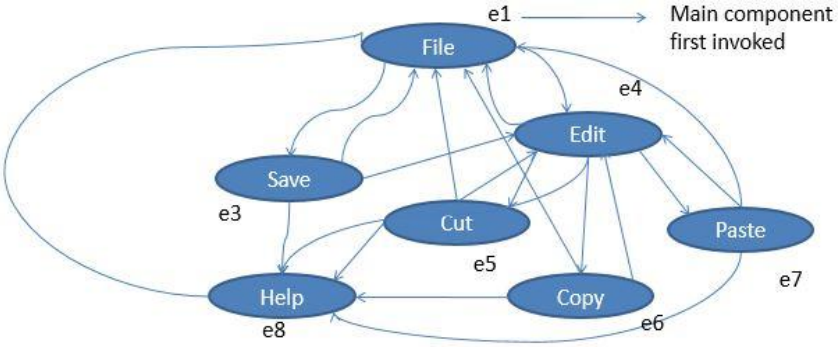

Fig.5: May- Follow relationship representation graph

Table 2: May-Follow Relationship

Event	Path to event
e1	<e2, e3, e4, e8>
e3	<e1, e4, e8>
e4	<e1, e5, e6, e7, e8>
e5	<e1, e4, e8>
e6	<e1, e4, e8>
e7	<e1, e4, e8>

B. Dominates relationship:

Dominates edge relationship consist of single incoming edge from vertex x to vertex y and vertex y must be preceded by after execution of vertex x.

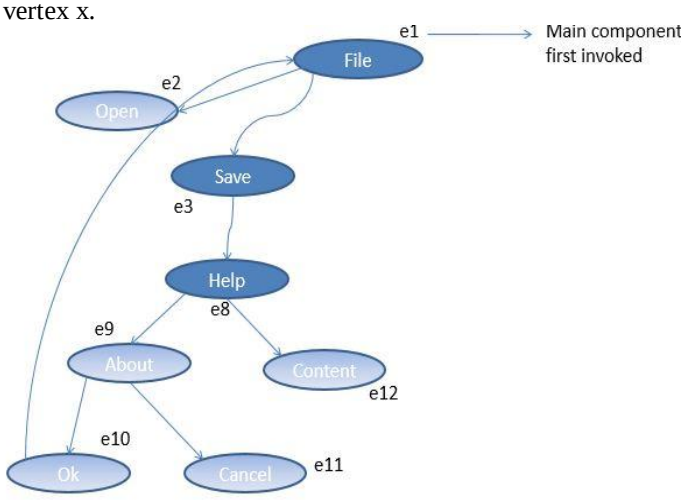

Fig.6: Dominates relationship Representation

Table 3: Dominates Relationship

Event	Path to event
e1	e10
e2	e1
e3	e1
e8	e3
e8	e3
e9	e8
e10	e9
e11	e9
e12	e8

STAGE 2: MAPPING

During ripping we know the location of each widget (eg: Dialog, window, menu) and the class that was used to create it. These attributes are address of each widget.GUI events are searched in the mapping from the ripper. If a match is not found a NULL entry is created.

S_0= {(File, Class, Menu);
(File, Enabled, True);
(Edit, Class, Menu);
(Edit, Enabled, True);
(Help, Class, Menu);
(Help, Enabled, True);
..
}

S_1= {(New, Class, Select);
(New, Enabled, True);
(Open, Class, Select);
(Open, Enabled, True);
..
}

S_2= {(Cut, Class, Select);
(Copy, Class, Select);
(Paste, Class, Select);
..
}
S_3= {(About, Enabled, True);
(Contents, Enabled, True);
..
}

a. Operator Event Mapping:

In order to keep the correspondence between original GUI events this contains operator name, type, modal dialogs and GUI events.

Two types of operator in events:

1. *System-Interaction Operators:*
   To represent sequences of GUI actions that a user might perform to eventually interact with the underlying software. A sequence of zero or more unrestricted-focus events is called as system-interaction event.
   Example:
   Edit_Cut = <Edit, Cut>
   Edit_Paste = <Edit, Paste>

2. *Abstract Operators:*
   Created from the restricted-focus events, which contain two parts:
   a. The ***prefix***of an abstract operator is the sequence of unrestricted-focus events that lead to restricted-focus event.
   b. The ***suffix***of an abstract operator represents the restricted-focus user interaction.

Table 4: Events Mapping Table

OPERATOR NAME	**TYPE/RELATIONSHIP**	**MODAL DIALOGS**	**GUI EVENTS**
FILE_NEW	System Interaction/ May-follow	Unrestricted focal events/ Modeless	<file, new>
FILE_OPEN	Abstract/Dominates	Restricted/modal	<file, open>
FILE_SAVE	System Interaction/ May-follow	Unrestricted focal events/ Modeless	<file, save>
EDIT_CUT			<edit, cut>
EDIT_COPY			<edit, cut>
HELP_ABOUT	Abstract/Dominates	Restricted/modal	<help, about>
HELP_CONTENTS			<help, contents>

STAGE 3: REPAIRING

Each statement is mapped to GUI events in EFG model or NULL after mapping. Simultaneously GUI widget referenced in checkpoints in scripts mapped to model level or NULL. Now GUI test script is ready to start repairing.

Given an EFG G1 and a mapped script which represents sequence of events(e1,e2...e12) and checkpoint(c1,c2...c12) where $e_i \in G1 U\{NULL\}$ *and* $c_i \in G1 U \{NULL\}$, the script needs repair if one of the following two conditions is satisfied.

Case 1: If the sequence having missing event which represents vertex at least one of event *ei*in the sequence is NULL.

Step 1: Algorithm procedure starts with the input test suite TS_0 automatically obtained EFG G1 for application A1.

Step 2: Test script TS1 which has null events in the sequence and represents the sequence of test script statements (s_1, s_2....s_n) which belongs to test suite.

Step 3: Event set and checkpoint set which has null events and checks every statement in the test case.

Step 4: Maps every statement to event e_i and checkpoint c_i then add to the event set and checkpoint set.

Step 5: For all event e_i and checkpoints set ci belongs to event set and check point set.

Step 6: If event e_i or c_i is NULL which is a missing event

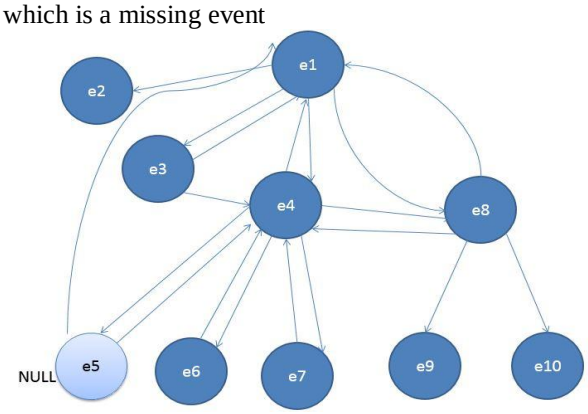

Fig.7: Missing event Representation of EFG

Step 7: let i=5 then

Step 8: repair event and update model (s_i,e_{i-1},e_{i+1}) //may-follow

Repair event = (s_5, e_{5-1}, e_{5+1})

= (s_5, e_4, e_6)

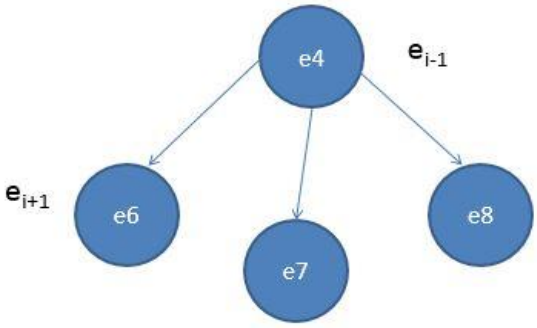

Fig.8: Repaired Event Representation of EFG

Step 9: Confirm script S_5;

Step 10: Add S_5 or e_5^1 to vertex as e_5

Step 11: add (e_{5-1}, e_5) and (e_5, e_{5+1}) to E_{may}

Step 12: add (e_4, e_5) and (e_5, e_6)

Step 13: Return (e_4, e_5, e_6)

Step 14: replace (e_4, e_5, e_6) as a repaired segment

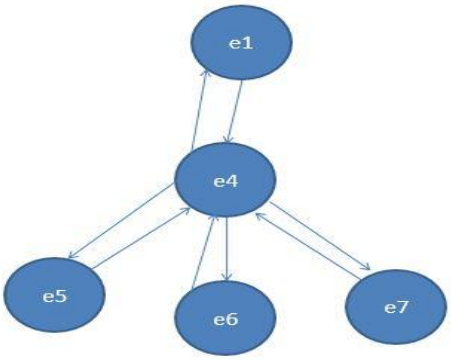

Fig.9:Repaired event Mapping Representation of EFG

Step 15: In order to delete a node from the event sequence then
Step 16: add (e_{5-1}, e_{5+1}) to E_{may}
Step 17: add and return e_4, e_6
Step 18: Return TS1

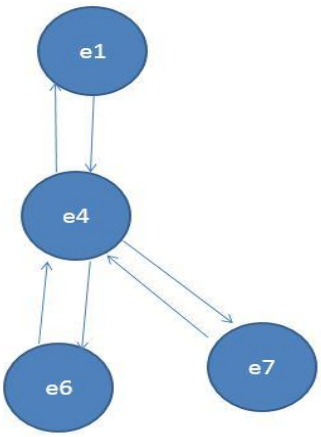

Fig.10: Deleted Event (e5) from EFG

Case 2: If the sequence of events having at least one pair (e_i,e_{i+1}) of adjacent events in sequence is not a valid edge in EFG.
Step 19: repair edge and update model (e_{i-1}, e_{i+1})// dominates
Step 20: let i=9 is missing edge

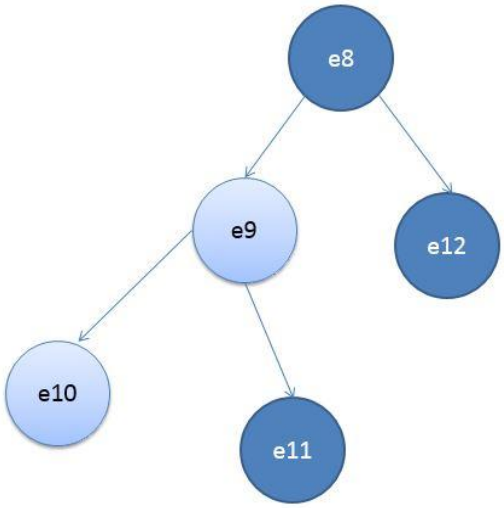

Fig.11: Missing edge Representation of EFG

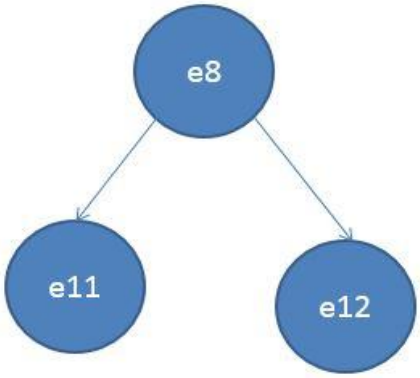

Fig.12: EFG after repairing missing edge

Step 21: repaired segment (e_{8-1}, e_{9+1})// e_9, e_{10}.
Step 22: repaired segment = (e_9, e_{10})
Step 23: replace e_9, e_{10} as repaired segment

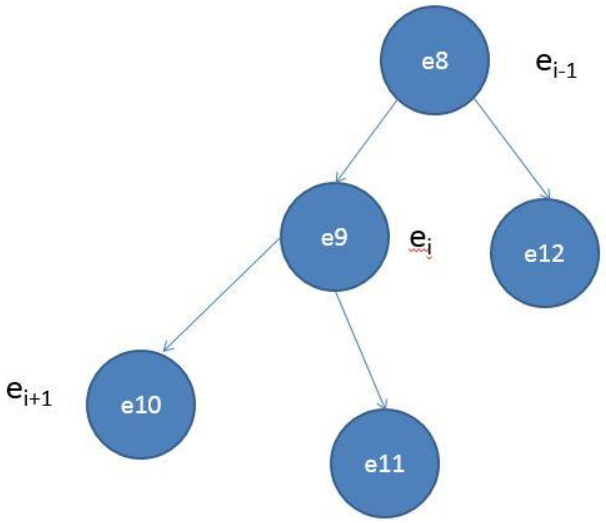

Fig.13:Final EFG after repairing all events

Step 24: Return TS1

V. RESULTS

The study process starts by selecting the two subject programs with two versions each and the team of testers will manually create a number of test scripts. Another team of testers uses a GUI test script repair in order to repair the original test scripts. The characteristics of study subject's withtwo different versions that can add, delete and modify the widgets are shown in table5.

Table 5: Characteristics of Study subjects

Subjects	All widgets	Modified widgets			Events	LOC
		Changed	Added	Deleted		
SP1.0	35				45	250
SP1.1	46	3	8	0	93	320
SP2.0	57				54	352
SP2.1	65	2	12	0	65	453

The process of test script with an engineered process examines that all the scripts that manually discarded are few which contains no checkpoints. The maximum and minimum average LOC of test scripts are shown in table 6.

Table 6: Size (LOC) of test scripts

Subjects	Min	Max	Avg
SP1	2	150	26
SP2	3	34	32

By computing the scripts of edit distances between each pair of events shows diversity and obtain the average distance and its ratio to average test length. The results are shown in table 7. The average edit distance of event sequence is greater than average length of test cases.

Table 7: Similarity of event sequences of test cases

Subjects	Avg.Dist	Avg.Len	Ratio
SP1.0	23.3	27.8	1.27
SP1.1	18.7	16.2	1.10

The average similarities of three applications are 0.16, 0.21. Jaccard index of two applications shows a reasonable degree of diversity in our test scripts.

Table 8: Similarity of code coverage of test cases

Subjects	Min	Max	Avg
SP1	0.13	1.00	0.16
SP2	0.63	1.00	0.21

The code and event coverage of the scripts is reasonably high with respect to test scripts. Coverage of original test suites is quite close to the coverage of repaired test suites.

Table 9: Coverage of Regression Test Scripts

Coverage(%)	SP2.0	SP2.1
Code	68.3	51.2
Event	59.8	68.7

VI. CONCLUSION AND FUTURE WORK

GUI test script repair in regression testing is presented a test script repair technique for GUI Application. First we generate test cases for GUI Application by using Ripping and Mapping techniques after generating test cases we run on selenium tool to check whether the Application working properly or not. After this, if we have done any modifications in GUI Application the test cases becomes NULL or formed unusable test cases for avoiding this problem we created a test script repair technique.

To improve the experiment, first the test script repair process proposed the study of GUI applications across multiple versions. Given that the test scripts for version 1 of an application, such that every application must have preceded version 0 and the later version 2. The extension of test script repairs from versions which can trace the evolution of test scripts across multiple versions v0 to v1 and v1 to v2 examine the ability to test functionality. The effects of size and change on quality and cost for the repair are also considered as a research question. If test script is done on each commit we can maintain quality for the small modifications will help to improve the EFG model and mapping tables.

REFERENCES

[1] Emily Kowalczyk, Atif Memon, "*Extending Manual GUI Testing beyond Defects by Building Mental Models of the Software Behaviour*," IEEE/ACM International Conference on Automated Software Engineering Workshop., p.p.35-41, 2015.

[2] Domenico Amalfitano, Porfirio Tramontana, Nicola Amatucci, Anna Rita Fasolino, "*A Conceptual Framework for the Comparison of a Fully Automated GUI Testing methods*,"IEEE/ACM International Conference on Automated Software Engineering Workshop, p.p.50-57, 2015.

[3] Eman M. Saleh,Omar Al Sheik Salem, "*A Model Driven Engineering Transition Based GUI Testing Techniques*," International Conference on Computational Science and Computational Intelligence, p.p.108-113., 2015.

[4] Hsiang-Lin Wen, Tzong-Han Hsieh, Cheng-Zen Yang, and Chia-Hui Lin, "*PATS: A Parallel GUI Testing Framework for an Android Applications*,"IEEE 39th Annual International Computers, Software & Applications Conference, p.p.210-215, 2015.

[5] Haowen Zhu, Xiaojun Zhang, Ke Shen, and Xiaojun Ye, "A Context-aware Approach for a Dynamic GUI Testing of an Android Applications," IEEE 39th Annual International Computers, Software & Applications Conference, p.p.248-253, 2015.

[6] Gennaro Imparato, "A Combined Technique of a GUI Ripping and Input Perturbation Testing for Android Apps," IEEE/ACM 37th IEEE International Conference on Software Engineering, p.p.760-762, 2015.

[7] Kevin Salvesen, Florian Gross, Gordon Fraser, Andreas Zeller, and Juan P. Galeotti, "Using Dynamic Symbolic Execution to Generate Inputs in a Search-Based GUI Testing,"IEEE/ACM International Workshop on Search-Based Software Testing, p.p.32-35, 2015.

[8] Zhi-Wei He, Cheng-Gang Bai, "GUI Test Case Prioritization by a State-coverage Criterion,"IEEE/ACM 10th International Workshop on Automation of Software Test, p.p.18-22., 2015.

[9] Xun Yuan, Myra B. Cohen, "GUI Interaction Testing: Incorporating Event Context,"IEEE Transaction on Software Engineering, Vol. 37, No. 4, p.p.559-574., July/August 2011.

[10] Emil Al´ egroth,Rafael A.P. Oliveira,Zebao Gao, and Atif Memon, "Conceptualization and Evaluation of a Component-based Testing Unified with Visual GUI Testing: an Empirical Study," IEEE Conference paper, 2015.

[11] Chien-Hung Liu , Shan-Jen Cheng, Yung-Chia Hsiao, xeng-Ming Chu, and Yung-Chia Hsiao, "Capture and Replay Testing for an Android Applications,"International Symposium on Computer, Consumer and Control, p.p.1129-1132, 2014.

[12] Lan Lin, Jia He, and Feng guang Song, "Usage Modelling through Sequence Enumeration for Automated Statistical Testing of a GUI Application," IEEE Conference Paper, p.p.82-85, 2014.

[13] Mingsong Chen, Zishan Qin, "An AD Based on Automated GUI Testing Framework for an Smartphone Applications,"Eighth International Conference on Software Security and Reliability, p.p.68-77, 2014.

[14] Harvinder Kaur,Puneet Jai Kaur, "A GUI Testing based Unit Testing Techniques for an Anti – pattern Based Identification," IEEE Conference Paper, p.p.779-782, 2014.

[15] Ali Darvish, Carl K. Chang, "An Automatic Discovery and Validations of a State based GUI Invariants," IEEE 38th Annual International Computers, Software and Applications Conference, p.p.65-74, 2014.

[16] Ali Darvish, Carl K. Chang, "Black - box Test case Data Generation for An GUI Testing," International Conference on Quality Software, p.p.133-138, 2014.

[17] Miguel Nabuco,Ana C. R. Paiva, and Pedro Costa,"PBGT for An Mobile Applications," International Conference on the Quality of Information and Communications Technology, p.p.66-74, 2014.
[18] M. Moreira, Rodrigo M. L, and Ana C. R. Paiva, "A GUI Modeling Domain Specific Language for an Pattern-Based GUI Testing by using PARADIGM," IEEE Conference Paper, 2014.
[19] Bao N. Nguyen, Bryan Robbins, and Atif Memon, "The First Decade of a GUI Ripping: Extensions, Applications, and Broader Impacts," p.p.11-20, 2013.
[20] GrischaLiebel, Emil Al egroth, and Robert Feldt, "State of Practice in a GUI-based System and Acceptance Testing: An Industrial Multiple Case Study," Euromicro Conference Series on Software Engineering and Advanced Applications, p.p.17-24, 2013.
[21] Om Prakash Sangwan, TulsiKushwaha, "Prediction of a Usability Level of Test Cases for GUI Based Applications Using Fuzzy Logic," IEEE Conference Paper., p.p.83-86, 2013.
[22] Pekka Aho NadjaMenz, and Tomi Räty, "Dynamic Reverse Engineering of a GUI Models for Testing," IEEE Conference Paper, p.p.441-447, 2013.
[23] Jing Feng, Bei-Bei Yin, Kai-Yuan Cai, Zhong-Xing Yu, "3-way GUI Test CasesBased on Event-wise Partitioning," IEEE International Conference on Quality Software, p.p.89-97, 2012.
[24] Gagandeep, Dr. JyotsnaSengupta, "Automatic Generation of Regression Test Cases for Web Components using Domain Analysis and Modeling," International Journal of Computer Applications (0975 – 8887),Volume 11– No.12,P.P.14-17, December 2010.
[25] Datchayani M, Arockia Xavier Annie R, and Yogesh P, "Test Case Generation and Reusing Test Cases for GUI Designed with HTML," JOURNAL OF SOFTWARE, VOL. 7, NO. 10, P.P.2269-2277, OCTOBER 2012.
[26] ATIF M. MEMON, "Automatically Repairing Event Sequence-Based GUI Test Suites for Regression Testing," ACM Transactions on Software Engineering and Methodology, Vol. 18, No. 2, Article 4, Pub. date: October 2008.
[27] Zhongxing Yu, Matias Martinez, Benjamin Danglot, Thomas Durieux and Martin Monperrus, "Test Case Generation for Program Repair: A Study of Feasibility and Effectiveness," International journal of software testing, P.P.1-12,1 march 2017.
[28] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba, "Test Case Generation and Test Data Extraction Techniques," International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 11 No: 03,p.p.82-89.
[29] Indumathi C P, Selvamani K, "Test Cases Prioritization using Open Dependency Structure Algorithm," International Conference on Intelligent Computing, Communication & Convergence (ICCC),p.p.250-255,2015.
[30] Bestoun S. Ahmed, Mouayad A. Sahib, Moayad Y. Potrus, "Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing," Engineering Science and Technology an International Journal 17 (2014) p.p.218-226.

