A Study of Testing Pitfalls in Graphical User Interface Testing and Its Elucidation

Iqra Tariq, Tasleem Mustafa

Abstract— The process of testing a product's interface is called graphical user interface testing conceded out to make certain that it meets its written specifications. Testing includes booming set of errands and evaluating the result with the estimated output and capability to recur equivalent set of chores several times. It includes different data inputs but same level of accuracy. GUI's provide large interaction space to the users. During testing, it is important to "adequately cover" this interaction space. Implementing GUI testing in earlier phases of the software development life cycle improves excellence, speeds up progress process and reduces risks towards the end product. Manual selection requires a tedious code inspection and does not scale. This research will uncover factors that affect product testing, and will evaluate them with fault related factors in GUI testing. The approach is to provide elucidation for these testing pitfalls.

Index Terms—Automation, D&C, Elucidation, GUI Testing, Interface, Pitfalls, Testing,

1 INTRODUCTION

A PC interface is said to be versatile on the off chance that it enhances the connection with every individual client. Basic retention of such connection is not adequate. Changes ought to additionally be delivered from the speculation of past experience and lead to new associations. An interface is a common limit crosswise over which autonomous frameworks correspond with one another. Interface permits making a proper physical association. With the goal that correspondence should be possible successfully. In software engineering and human-PC association, the client interface (of a PC project) alludes to the graphical, literary and sound-related data the system presents to the client. The client utilizes a few control groupings, (for example, keystrokes with the PC console, developments of the PC mouse, or choices with the touchscreen) to control the project. There are a few various types of client interfaces among them the most imperative are:

1.1 COMMAND LINE INTERFACE

A CLI (summon line interface) is a client interface to a PC's working framework or an application in which the client reacts to a visual brief by writing in an order on a predetermined line, gets a reaction once again from the framework, and after that enters another charge, et cetera. The MS-DOS Prompt application in a Windows working framework is a case of the procurement of a summon line interface. Today, most clients incline toward the graphical client interface (GUI) offered by Windows, Mac OS and others. Ordinarily, a large portion of today's UNIX-based frameworks offer both an order line interface and a graphical client interface. CLIs are frequently utilized by developers and framework directors, in building and logical situations, and by actually propelled PC clients. CLIs are likewise mainstream among individuals with visual handicap, subsequent to the summons and reactions can be shown utilizing Refreshable Braille shows.

1.2 GRAPHICAL USER INTERFACE

It is a kind of interface that permits clients to associate with electronic gadgets through graphical symbols and visual markers, for example, auxiliary documentation, instead of content based interfaces, wrote summon names or content route. GUIs were acquainted in response with the apparent steep expectation to learn and adapt of order line interfaces (CLIs), which oblige charges to be written on the console.

The activities in a GUI are generally performed through direct control of the graphical components. Notwithstanding PCs, GUIs can be found close by held gadgets, for example, MP3 players, convenient media players, gaming gadgets and littler family, office and industry gear. The expression "GUI" tends not to be connected to other low-determination sorts of interfaces with presentation resolutions, for example, computer games (where HUD is favored), or not confined to level screens, as volumetric showcases on the grounds that the term is limited to the extent of two-dimensional showcase screens ready to depict nonexclusive data, in the PC's convention science research at the PARC (Palo Alto Research Center). A GUI may be intended for the necessities of a vertical business sector as application-particular graphical client interfaces. Illustrations of utilization particular GUIs incorporate mechanized teller machines (ATM), purpose Of-Sale touchscreens at eateries, self-administration checkouts utilized as a part of a retail location, aircraft self-ticketing and registration, data stands in an open space, similar to a train station or an exhibition hall, and screens or control screens in an implanted mechanical application which utilize a constant working framework (RTOS).

1.3 TESTING:

Testing is the act of making target judgments in regards to the degree to which the framework (gadget) meets, surpasses or neglects to meet expressed destinations. It includes the execution of a product part or framework segment to assess one or more properties of hobby. By and large, these properties demonstrate the degree to which the segment or framework under test:

- meets the necessities that guided its configuration and improvement,
- responds effectively to a wide range of inputs,
- performs its capacities inside of a satisfactory time,
- is adequately usable,
- can be introduced and keep running in its proposed surroundings

- *Iqra Tariq is currently pursuing MS degree program in computer science in University of Agriculture, Faisalabad, Punjab, Pakistan , PH-00923006673700 E-mail: iqratariq@yahoo.com*
- *Tasleem Mustafa is currently working as Chairman in Department of Computer science at University of Agriculture, Faisalabad, Punjab, Pakistan. E-mail:tasleemustafa@uaf.edu.pk*

• Achieves the general result its partner's yearning.

1.4 Purpose of testing:

There are two crucial purposes of testing: confirming acquisition details and overseeing danger. To begin with, testing speaks the truth checking that what was determined is what was conveyed: it confirms that the item (framework) meets the practical, execution, configuration, and usage necessities recognized in the acquirement determinations. Second, testing speaks the truth overseeing danger for both the obtaining office and the framework's seller/designer/integrator. The testing project is utilized to distinguish when the work has been "finished" so that the agreement can be shut, the merchant paid, and the framework moved by the office into the guarantee and upkeep period of the undertaking.

2 GUI testing

Regardless of its significance, the advancement group has been moderate to incorporate GUI testing as a center practice, principally on the grounds that GUI testing is troublesome. In this paper we first acquaint suggestions and practices with compose and keep up vigorous GUI tests in a smooth, brisk and common way.

GUI testing is the procedure of guaranteeing fitting usefulness of the graphical client interface (GUI) for a given application and verifying it fits in with its composed particulars.

Notwithstanding usefulness, GUI testing assesses plan components, for example, design, hues, textual styles, text dimensions, marks, content boxes, content arranging, inscriptions, catches, records, symbols, connections and substance. GUI testing procedures can be either manual or programmed, and are regularly performed by third - gathering organizations, as opposed to engineers or end clients.

GUI testing can oblige a ton of programming and is tedious whether manual or programmed. Typically the product writer works out the planned capacity of a menu or graphical catch for clarity so that the analyzer won't be befuddled as to the normal result. GUI testing likewise tends to test for certain project practices that clients expect, similar to an hourglass when the system is occupied, the F1 key raising the help framework and numerous other basic points of interest.

2.1 Troubleshooting GUI test failures

We consider that on numerous events investigating GUI tests disappointments may require more exertion that written work a test. All in all, GUI test disappointments are because of these basic reasons:

• Environmental conditions
• A GUI segment couldn't be found
• More than one GUI segment coordinated the given inquiry criteria.

2.2 GUI Testing Steps

Despite the fact that GUIs have attributes, for example, client occasions for info and graphical yield, that are not quite the same as those of customary programming and in this manner require the advancement of distinctive testing systems, the general procedure of testing GUIs is like that of testing ordinary programming. The testing strides for customary programming, stretched out for GUIs, take after:

Determine what to test

Amid this first stride of testing, scope criteria, which are sets of guidelines used to figure out what to test in programming, are utilized. In GUIs, a scope standard may oblige that every occasion been executed to figure out if it acts effectively.

Generate test input

The test info is a critical piece of the experiment and is built from the product's determinations and/or from the product's structure. For GUIs, the test data comprises of occasions, for example, mouse clicks, menu choices, and item control activities.

Generate Predictable Output

Test prophets produce the regular yield, which is utilized to outline out if or not the product executed efficiently amid testing. A test forecaster is a gadget that figures out if or not the give way from the product is similar to the usual yield. In GUIs, the ordinary yield incorporate screen depictions and locations and titles of windows.

Implement Test Cases And Authenticate Output

Experimentation is executed on the product and its yield is contrasted and the normal yield. Completing of the GUI's experiment is completed by performing all the info occasions indicated in the experiment and complementary the GUI's yield with the usual give way as given by the test forecasters.

Conclude If The GUI Was Sufficiently Tested

When all the tests have been executed on the actualized programming, the product is broke down to check which of its pieces were really tried. In GUIs, such an examination is expected to distinguish the occasions and the subsequent GUI states that were tried and those that were missed. Note that this stride is necessary on the grounds that it might not usually be imaginable to test in a GUI finishing what is needed by the range criteria. In the wake of testing, issues are distinguished in the product and remedied. Alterations then prompt relapse testing.

Execute Regression Testing

Relapse testing is exploited to assist guarantee the correctness of the altered parts of the product and in addition to build up confidence that progressions have not adversely influenced previously tried parts. A deterioration test suite is created that includes of:

(1) A different subset experiments to retest parts of the first programming that may have been influenced by alterations.
(2) New experiments to test influenced parts of the product, not tried by the chose experiments. In GUIs, relapse testing includes investigating the progressions to the design of GUI articles, selecting experiments that ought to be rerun, and additionally creating new experiments.

Any GUI testing technique must perform the greater part of the above steps. As of now, GUI test fashioners regularly depend on record/playback instruments to test GUIs. The procedure included in utilizing these apparatuses is to a great extent manual, making GUI testing moderate and lavish. Every one of the systems must be coordinated, utilizing a typical representation so that aftereffects of one apparatus are good with the others.

• The GUI testing errands ought to be as computerized as could be allowed so that the test architect's Work is rearranged.
• The general testing cycle characterized by the methods ought to be proficient since programming testing is typically a repetitive and extravagant procedure. Wastefulness may prompt disappointment and relinquishment of the methods.
• The procedures ought to be powerful. At whatever point the GUI enters a surprising express, the testing calculations ought to identify the slip state and report all data important to investigate the GUI.
• The instruments/procedures ought to be versatile. Test data (e.g., experiments, prophet data, scope report, and slip report) produced and/or gathered on one stage ought to be usable on every other stage on which the GUI can be executed.
• Finally, the strategies ought to be sufficiently general to be appropriate to an extensive variety of GUIs.

A Graphical User Interface (GUI) is a graphical front-end to a product framework. A GUI contains graphical items with certain particular qualities which can be utilized to focus the condition of the GUI whenever. Programming creating associations dependably craving to test the product altogether to get greatest certainty about its quality. Yet, this requires massive push to test a GUI application because of the unpredictability included in such applications. This issue has prompted the mechanization of GUI testing and diverse procedures have been proposed for computerized GUI Testing.

2.3 Approaches of GUI Testing

It is possible through these ways:

Manual Based Testing

Underneath this methodology, graphical displays are checked physically by analyzers in conformance with the necessities expressed in business documents.

Record and Replay

GUI testing should be possible utilizing mechanization devices. This is done in two sections. Amid Record, test steps are caught into the computerization device. Amid playback, the recorded tests procedures are executed on the Application under Test. A catch replay instrument is an arrangement of programming projects that catch client inputs and stores it into an organization (a script) suitable to be utilized at a later time to replay the client inputs. A catch/playback device that bolster the accompanying capacities could be utilized as a part of a more proficient and completely coordinated test improvement environment.

• record scripts of client/framework connections
• user access to scripts for altering/upkeep
• user capacity to embed acceptance orders in the script
• allows replay of the recorded

Model Based Testing

A model is a graphical depiction of framework's conduct. It helps us to comprehend and anticipate the framework conduct. Models help in an era of proficient experiments utilizing the framework necessities. Taking after should be considered for this model based testing:

• Build the model
• Determine Inputs for the model
• Calculate expected yield for the model
• Run the tests
• Compare the real yield with the normal yield
• Decision on further activity

Acceptance testing with "GUI Test Drivers"

GUI test driver devices help the designer do utilitarian/acknowledgment testing through a client interface, for example, a local GUI or web interface. Table-based acknowledgment testing. Beginning from a client story (or utilization case or literary necessity), the client enters in a table the program's desires conduct.

REGRESSION TESTING

Our relapse testing procedure comprises of two sections: a checker that sorts an experiment as being usable or unusable; if unusable, it additionally figures out whether the experiment can be repaired. The second part is the repairer that repairs the unusable, repairable experiment. In spite of the fact that for simplicity of clarification, these two sections are dealt with exclusively, they could be consolidated together in a usage. The relapse analyzer takes as information the G-CFGs and G-call trees for both the first and changed GUI, the legitimate beginning states SI for the altered GUI, and experiments for the first GUI. The checker parcels the first test suite into unusable and usable experiments. Imperatively, it can likewise figure out if or not an unusable test can be repaired. Naturally, an experiment can be repaired if its starting state is still legitimate for the adjusted GUI (i.e., the GUI can be brought into the state) and if its occasion arrangement can be made lawful for the altered GUI. To make a GUI occasion arrangement legitimate, we acquire a lapse recuperation strategy from compiler innovation; we skip occasions or attempt to embed a solitary new occasion until a lawful occasion grouping is gotten. This arrangement can be found by including so as to skirt occasions or occasions from the changed GUI.

Event capture

To battle this and different issues, analyzers have gone 'in the engine' and gathered GUI collaboration information from the basic windowing framework. By catching the window "occasions" into logs the communications with the framework are currently in an organization that is decoupled from the presence of the GUI. Presently, just the occasion streams are caught. There is some sifting of the occasion streams important since the surges of occasions are typically extremely point by point and most occasions aren't straightforwardly applicable to the issue. This methodology can be made simpler by utilizing a MVC construction modeling for instance and making the perspective (i.e. the GUI here) as straightforward as could be expected under the circumstances while the model and the controller hold all the rationale. Another methodology is to utilize the product's implicit assistive innovation, to utilize a HTML interface or a three-level structural planning that improves it likewise conceivable to partitioned the client interface from whatever is left of the application.

Another approach to run tests on a GUI is to incorporate a driver with the GUI so that orders or occasions can be sent to the product from another project. This strategy for straightforwardly sending occasions to and getting occasions from a framework is exceptionally attractive when

testing, subsequent to the data and yield testing can be completely computerized and client blunder is disposed of.

HUMAN TESTING

One way to deal with GUI testing is to just have a human analyzer perform an arrangement of client operations on the objective application and confirm that it is acting effectively. In any case, this manual methodology can be tedious, dull, and blunder inclined. A more effective methodology is to compose your GUI tests such that client activities are performed in a mechanized way. The mechanized methodology permits you to run your tests rapidly and dependably in a repeatable way.

COMPUTERIZED GUI TESTING

Modernized GUI Testing is a response for each one of the issues raised with Manual GUI Testing. An Automated GUI Testing mechanical assembly can playback all the recorded plan of assignments, difference the results of execution and the typical lead and report accomplishment or failure to the test modelers. Once the GUI tests are made they can without quite a bit of a stretch be repeated for different number of times with particular data sets and can be contacted cover additional components at a later time.

EVENT-FLOW MODEL

The occasion stream model contains two sections. The principal part encodes every occasion as far as preconditions, i.e. the state in which the occasion may be executed, and impacts, i.e. the progressions to the state after the occasion has executed. The second part speaks to every conceivable arrangement of occasions that can be executed on the GUI as an arrangement of coordinated diagrams.

COMPONENT LOOKUP

To reenact a client cooperating with a GUI, we initially need to get a reference on a GUI part. FEST does not force a particular method for performing segment lookups. Rather, it offers the accompanying lookup sorts: by segment name, by segment sort, and by client characterized hunt criteria.

BY COMPONENT NAME

As beforehand expressed, utilizing a remarkable name or identifier for GUI segments ensures that we can simply discover them, paying little mind to any adjustment in the GUI

BY COMPONENT TYPE

FEST likewise gives segment lookup by sort. This sort of lookup is dependable the length of the GUI under test has one and only segment of such sort generally FEST won't know which GUI part is the one we are keen on, and the test will come up short.

An ever-present means of communicating with software systems is graphical user interfaces. GUI acts as a front end to the core application code and also responds to user procedures such as click event or menu selection. The interaction between core code and GUI is done by method calls. With the help of GUI software's are becoming more users friendly. A large portion of code is dedicated for GUI.GUI comprises as much as 60 percent of total code. Due to increasing importance of GUIs its testing has become vital. It enhances entire system's safety, sturdiness, and usability [1].

The prevalent use of GUIs for interacting with software is leading to the erection of more and more complex GUIs. With the increasing complexity come challenges in testing the appropriateness of a GUI and the core software. The complexity of modern GUIs has made its testing more and more difficult task [2]. Graphical user interfaces (GUIs) provides an enormous number of impending event sequences to users. GUI's are elastic in nature. Therefore the number of event sequences increases exponentially with length. An ever-present challenge of GUI testing is to make those sequences that lead to potentially challenging situation.

GUI testing techniques currently used are deficient and mostly manual. The most common technique used in GUI testing is record-playback. A test designer generates mouse and keyboard events and interact GUI by means of it. The record tool captures interfaces, GUI session, user events and stores them. Later on the tester playback these recorded sessions to retest the events with different inputs. The pitfall of this process is its exceptionally labor intensive and relies on tester skills [3].

Higher level of efficiency can achieved by automatic test case generator but it needs programmer to code all possible potential decision points. Most of the important GUI decisions are missed by the record-playback technique. To avoid this beta copies of software are released. Software testing is exhaustive, laborious a resource consuming. Traditional software testing techniques do not effectively tackle GUI testing pitfalls.

3 REVIEW OF LITERATURE

Kim and Yoon[4] characterized that scheming the client interfaces of current gadgets and programming projects is complimenting an additional multifaceted assignment in view of the rising requests of execution and straightforwardness. The modeler ought to build up the correspondence among the clients and the interfaces considering client goals, needs, capacities and realistic interface ways. One more reason for complexities is the shortened time term for mounting crisp items. Draftsmen are obliged to incline toward outline systems that are all around sorted out and predictable for planning worthy client interfaces.

Maxion and Reeder [5] Security may be traded off when people commit errors at the client interface. Clear content is erroneously sent to reporters, delicate records are left unprotected, and mistakenly designed frameworks are left defenseless against assailants. Such oversights may be faulted for human lapse, yet the normality of human blunder recommends that slip-ups may be preventable through better interface outline. Certain client interface develops drive clients toward lapse, while others encourage achievement. Two security-touchy client interfaces were assessed in a research facility client study. The Windows XP document consents interface and an option interface, called Salmon, composed as per a lapse maintaining a strategic distance from standard to check the deceptive builds in the XP interface. A logical hypothesis in its initial phases of improvement is displayed.

Bowen and Reeves[6] characterized that the improvement of client interface administration frameworks (UIMS) in view of the intelligent partition of framework usefulness and client interface (UI) is illustrated by the

Seeheim model. The division permits us to not just focus on the diverse apprehensions which distinctive measures of the framework advancement existing, at the same time, all the more vitally, takes into account diverse methodologies and outline procedures. They portrayed the presentation model, which formally catches a casual UI plan, and talked about how they utilized outlines as a part of a formal refinement process and in addition for configuration equality and consistency checking. The presentation model permitted catching static properties of a UI outline and in this manner indicated with formalism, FSM, to catch dynamic UI conduct in view of UI capacities which change the accessible Functionality of the UI for a client, giving PIMs.

Eslambolchilar et al. [7] depicted element frameworks approach for continuous communication of interfaces. It permits the utilization recreations, and efficient devices to architects for looking at the execution and strength of the framework. Element approach alongside manual control model aides in institutionalization of the framework. It likewise tunes up the framework parameters before the culmination.

Jimenez and Librane [8] expressed that the treatment of more mind boggling applications has lead to the advancement of apparatuses for UI outline in which the framework planner lives up to expectations with abnormal state and conceptual models, specifically visual displaying of UIs. The level of deliberation on UI configuration permits the UIs to be adjusted and created for a few stages. The outline of UIs is currently an improvement process in which an abnormal state detail and displaying of the UI assume a urgent part. They examined a Model-driven advancement (MDD) procedure for demonstrating WIMP (Windows, Icons, Menus and Pointers) WIMP-based UIs. We have characterized another sort of UML graph.

Yuan and Memon [9] expressed that past they added to an input based strategy to upgrade a two way covering test suite. They did this by breaking down the impact of each GUI occasion on the GUI's run-time state and getting sets of occasions that impact each other by they way they alter the GUI's state. This "impact" was caught as the Event Semantic Interaction (ESI) connection and displayed as a chart called the ESI Graph (ESIG). An imperative property of these experiments is that every adjoining occasion are connected by means of the ESI relationship. They abridged the diverse systems in their examination strategy. Be that as it may, albeit superior to the comprehensive approach, the quantity of experiments needed for the ESIG-based strategy additionally becomes exponentially with length for most applications, making it hard to test.

3.1 Test Principles Applied to GUIs

Our proposed approach to testing GUIs is guided by several principles, most of which should be familiar. By following these principles we will develop a test process which is generally applicable for testing any GUI application.
We intend to categorize errors into types and design test to detect each type of error in turn. In this way, we can focus the testing and eliminate duplication.

Layered and staged tests

We will organize the tests into a sequence of test stages. The belief here is that we bring tests of the bottom level of detail in constituents up front. We instrument integration tests of components and test the integrated application last. In this way, we can build the testing up in trusted layers.

Test automation...wherever possible

Automation furthermost habitually flops because of over-ambition. By piercing the test development into stages, we can seek and find opportunities to make use of automation where appropriate, rather than trying to use automation everywhere.

High Level Test Process

An outline test process is presented in Figure 1 - The high-level test process. We can split the process into three overall phases: Test Design, Test Preparation and Test Execution. In this paper, we are going to concentrate on the first stage: Test Design, and then look for opportunities for making effective use of automated tools to execute tests.

Types of GUI errors

We can list some of the multifarious errors that can occur in a client/server-based application that we might reasonably expect to be able to test for using the GUI. The list in Table 1 is certainly not complete, but it does demonstrate the wide variety error types. Many of these errors relate to the GUI, others relate to the underlying functionality or interfaces between the GUI application and other client/server components.

TABLE 1
BASIC GUI ERRORS

Data validation	**Correct window modality?**
Incorrect field defaults	**Window system commands not available/don't work**
Mis-handling of server process failures	**Control state alignment with state of data in window?**
Mandatory fields, not mandatory	**Focus on objects needing it?**
Wrong fields retrieved by queries	**Menu options align with state of data or application mode?**
Incorrect search criteria	**Action of menu commands aligns with**
Field order	**state of data in window**
Multiple database rows returned, single row expected	**Synchronization of window object content**
Currency of data on screens	**State of controls aligns with state of data in window?**
Window object/DB field correspondence	

3.2 Four Stages of GUI Testing

This paper proposes a GUI test design process that fits into an overall test process. Test design becomes a series of straightforward activities, each focusing on different types of error. The question now is, 'how does this fit into existing test processes?' To help readers map GUI test types to a more traditional test process, we have grouped the test types in four stages.
The four stages are summarized in Table 2 below. We can map the four test stages to traditional test stages as follows:

- Low level - maps to a unit test stage.
- Application - maps to a unit test or functional system

test.
- Integration - maps to a functional system test stage.
- Non-functional - maps to non-functional system test stage.

The mappings described above are approximate. Clearly there are occasions when some 'GUI integration testing' can be performed as part of a unit test. The test types in 'GUI application testing' are equally suitable in unit or system testing. In applying the proposed GUI test types, the objective of each test stage, the capabilities of developers and testers, the availability of test environment and tools all need to be taken into consideration before deciding whether and where each GUI test type is implemented in your test process. The GUI test types alone do not constitute a complete set of tests to be applied to a system. We have not included any code-based or structural testing, nor have we considered the need to conduct other integration tests or non-functional tests of performance, reliability and so on. Your test strategy should address all these issues.

TABLE 2
MAPPING OF TEST TYPES AND STAGES

Stage	Test Types
Low Level	Checklist testing Navigation
Application	Equivalence Partitioning Boundary Values Decision Tables State Transition Testing
Integration	Desktop Integration C/S Communications Synchronization
Non-Functional	Soak testing Compatibility testing Platform/environment

3.3 JUSTIFYING AUTOMATION

Automating test execution is normally justified based on the need to conduct functional regression tests. In organizations currently performing regression test manually, this case is easy to make - the tool will save testers time. However, most organizations do not conduct formal regression tests, and often compensate for this 'sub-consciously' by starting to test late in the project or by executing tests in which there is a large amount of duplication. In this situation, buying a tool to perform regression tests will not save time, because no time is being spent on regression testing in the first place. In organizations where development follows a RAD approach or where development is chaotic, regression testing is difficult to implement at all – software products may never be stable enough for a regression test to mature and be of value. Usually, the cost of developing and maintaining automated tests exceeds the value of finding regression errors.

We propose that by adopting a systematic approach to testing GUIs and using tools selectively for specific types of tests, tools can be used to find errors during the early test stages. That is, we can use tools to find errors pro-actively rather than repeating tests that didn't find bugs first time round to search for regression errors late in a project.

3.4 AUTOMATING GUI TESTS

Throughout the discussion of the various test types in the previous chapter, we have assumed that by designing tests with specific goals in mind, we will be in a better position to make successful choices on whether we automate tests or continue to execute them manually. Based on our experience of preparing automated tests and helping client organizations to implement GUI test running tools we offer some general recommendations concerning GUI test automation below.

TABLE 3
AUTOMATING GUI TESTS

• Pareto law	• We expect 80% of the benefit to derive from the automation of 20% of the tests.
• Hybrid Approach	• The tools to perform navigation and data entry prior to manual test execution. Consider using the tool for test running, but perform comparisons manually or 'off-line'.
• Coded scripts	• These work best for navigation and checklist-type scripts, where loops and case statements in code leverage simple scripts Are relatively easy to maintain as regression tests.
• Recorded Scripts	• Need to be customized to make repeatable. Sensitive to changes in the user interface.
• Test Integration	• Automated scripts need to be integrated into some form of test harness. Test harnesses are usually crude so custom built harnesses are required
• Migrating Manual Test Scripts	• Manual scripts document automated scripts Delay migration of manual scripts until the software is stable, and then reuse for regression tests.
• Non-Functional Tests	• Scripts can be reused for soak tests, but they must be of concern.Instrument these scripts to take response time measurements and re-use for performance testing.

Everyone on the world can make mistakes. Some of

them are of no importance but some of them become dangerous and expensive to bear. That's why everything made by human needs to be checked. GUI testing is really required to point out the defects and errors that were made during the development phases. It's essential since it makes sure of the Customer's reliability and their satisfaction in the application. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results. Testing is required for an effective performance of software application or product. GUI testing plays an important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.

4 Result and Discussion:

The intact field of human computer interaction is based on mental model research, which is based on the assumptions of how the users perceive systems and how the systems should be for better understanding of usable systems. The insight in user's cognitive processes when using a system could successfully change the current system and methods of designing interfaces. Designers are sometimes asked to become psychologists and psychologists are sometimes asked to become designers.

A survey was conducted to collect views of various testers from different software houses about the expected elucidations of the pitfalls found in GUI testing strategy. Such as, built in dependency, reduction in system call, code instrumentation, unsolicited events instrumentation, functionality in one function call, avoid dependencies between objects and documentation for every possible event, these were the expected elucidation for the pitfalls discussed in research. The results of the survey were presented in graphical form as below:

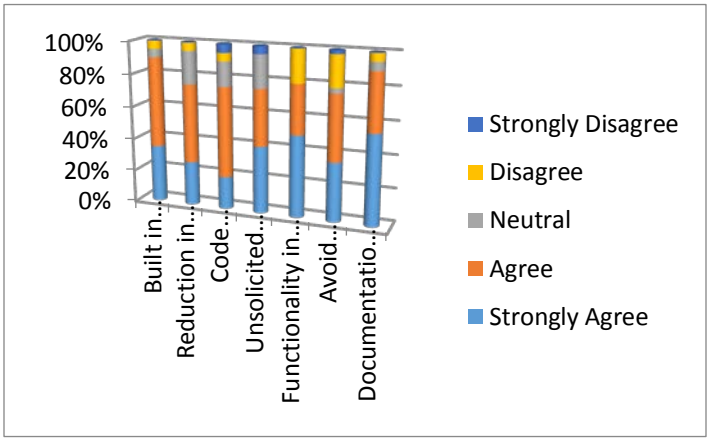

Fig 1: Survey Results

Round about 35% people were strongly agree about a solution of interface pitfall, built in dependency, 55% were agree, 5% were Neutral, 5% were disagree and 0% were strongly disagree. In response to second solution, "reduction in system command", 27% persons were strongly agree, 48% were Agree, 20% were neutral, 5% were disagree and only 0% were strongly disagree. Approximately, 20% people gave their consent as strongly agree, in the answer of third solution a of interface pitfalls, " code instrumentation" 55% as agree, 15% as neutral, 5% as disagree and 5% as strongly disagree. In response to " unsolicited events instrumentation" solution, it was recorded that 45% were strongly agree, 38% were agree, 22% were neutral, 0% were disagree, 5% were strongly disagree. Round about 0% people were agree with the notion of "functionality in one function call", 30% were agree, 0% were neutral, 20% were disagree and 0% were strongly disagree. 36% people belong to various software houses were strongly agree with the notion of " avoid dependencies between objects", 40% were agree, 3% were neutral, 19% were disagree and only 2% were strongly disagree. In response to a solution, " documentation for every possible event", 55% people recorded their idea as strongly agree, 35% as agree, 5% as neutral, 5% disagree and 0% as strongly disagree.

Future Work:

In this exploration more work to be done to refine the testing methodologies to transform them into convenient and adaptable GUI test technique. Our desire was to figure out GUI testing pitfalls that are a piece of framework test procedure. Mechanization regularly comes up short due to over-desire. By part the test procedure into stages, we can look for and discover chances to make utilization of computerization where suitable, as opposed to attempting to utilize robotization all over the place.

Refrences:

[1] Mijailović. Z and D. Milicev. 2014. Empirical analysis of GUI programming concerns. *International Journal of Human-Computer Studies,* 72:757-771.

[2] Yang. W., Z. Chen., Z. Gao., Y. Zou and X. Xu. 2014. GUI testing assisted by human knowledge: Random vs. functional. *The Journal of Systems and Software,* 89:76-86.

[3] Bae. G., G. Rothermel and D. Bae. 2014. Comparing model-based and dynamic event-extraction based GUI testing techniques: An empirical study. *The Journal of Systems and Software,*97:15-46.

[4] Kim, H and W. C. Yoon. 2005. Supporting the Cognitive Process of User Interface Design with Reusable Design Cases. International Journal of Human Computer Studies, 62:457-486.

[5] Maxion, R. A and R. W. Reeder. 2005. Improving user-interface dependability through mitigation of human error. International Journal of Human-Computer Studies, 63: 25-50

[6] Bowen. J and S. Reeves. 2007. Formal Models for Informal GUI Designs. Electronic Notes in Theoretical Computer Science. 183:57-72.

[7] Eslambolchilar. P. and R. M. Smith. 2008 Control centric approach in designing scrolling and zooming user interfaces. International Journal of Human-Computer Studies,66:838-856.

[8] Jimenez, J. M. A and L. Lribarne. 2008. An extension of UML for the modeling of WIMP user interfaces. Journal of Visual Languages and Computing, 19:695-720.

[9] Yuan. X and A. M. Memon. 2010. Iterative execution-feedback model-directed GUI testing. Information and Software Technology. 52:559-575.