1. Error handling

Verification and Error Handling

While implementing automated UI tests, tester has the following 3 options: ignore error-handling, implement error-handling, implement both exception and error handling. The specific of exception handling in automated testing is in a non-code fails: when application doesn't behave accordingly to the expected test flow.

In case when there is no error handling, a tester could have the following impediments:

- Whenever error occurred testing tool stops execution. Possibly a defect could be exposed at that point but 99% more likely it’s just a problem with the script
- Since at any point test execution could be broken scripts are set up and executed manually one by one. Test results are also validated manually. With all the manual involvement it becomes no longer really automated
- A continuous requirement going into the code means requirement keeping a code-familiar person along with the scripts. And knowledge transfer and learning curve problems in case of transitions
- Having low robustness and high maintenance cost scripts means keeping twice as expensive automated testers doing things neither better nor faster than manual testers

A set of verification and error handling are required to prevent those impediments.

GUI verification

Verification of GUI (if the button exists, or if the button is enabled before you want to click it or verify it's presence) before performing operations allows more accurate coverage and more robust test execution.

In order to implement this, a GUI error handling class can be created that will verify the element state.

Create a general StringBuffer that will collect the possible present errors, in a Setup class:

```
public static StringBuffer VERIFICATION_ERRORS = new StringBuffer();
```

Create an error handling class that contains multiple levels of verification, if the element is present, is enables, is not present, visible , based on the desired test to run:

```
public class PageErrorHandling {

/*verifies if the element is present or not on a page and waits a defined time for
that*/
public static Boolean verifyElementPresent(WebDriver webdriver, By locator, Boolean
screenshotRequired, Boolean isTestFailed) {

 /*Implement webdriver explicit wait that will wait for the specific element to be
available on page*/
 WebDriverWait wait = new WebDriverWait(webdriver, Setup.EXPLICIT_WAIT);
 try {
  wait.until(ExpectedConditions.presenceOfElementLocated(locator));
  return true;
 }
 catch (NoSuchElementException | NoSuchFrameException | NoSuchWindowException |
UnknownServerException | TimeoutException e) {
  /**method that handles if the screenshot should be made or not if the test fails/
  if (screenshotRequired){
   makeScreenshot();
   }
  /*method that handles if the test should be set as failed or not if the element is
not found. If true, the execution of the test will be stopped*/
  if (isTestFailed){
   test.fail();
  }
  /*gather all verification errors and add them in the string buffer*/
  Setup.VERIFICATION_ERRORS.append(
   "Element: " + locator+ " is not present on page \n -Caugth exception: "
   + e.getMessage() + "\n\n");
   return false;
 }
}
```

In order to use the method, you can call it in the main method that is interacting with the element:

```
public static void click(By locator){
 if (DriverErrors.verifyElementPresent(driver, locator, true, true)){
  driver.findElement(locator).click();}
}
```

Error handling

It is used in each single test step implementation and makes sure that each test verification or errors are handled and returned in the error logs. This can be at verifying if two variables are equal, or a specific statement is true or not.

```
/*method that creates the implementation of verify equals that checks if two string
values
* are equal and in case of error returns the step where the text did not match*/
public static void verifyEquals(String expectedString, String actualString, Boolean
screenshotRequired){
 try {
  assertEquals(expectedString,actualString);
 }
 catch (AssertionError ae) {
  //screenshot taken only if required in the step
  if (screenshotRequired) {
   makeScreenshot();
  }
  /*method that handles if the test should be set as failed or not if the specific
condition is not true. If true, the execution of the test will be stopped*/
  if (isTestFailed){
   test.fail();
  }
  Setup.VERIFICATION_ERRORS.append(
   "Current value is: "+actualString+ "but it should be"+expectedString+
   "\n-Caugth exception: "+ ae.getMessage() + "\n\n");
   }
 }
}
```