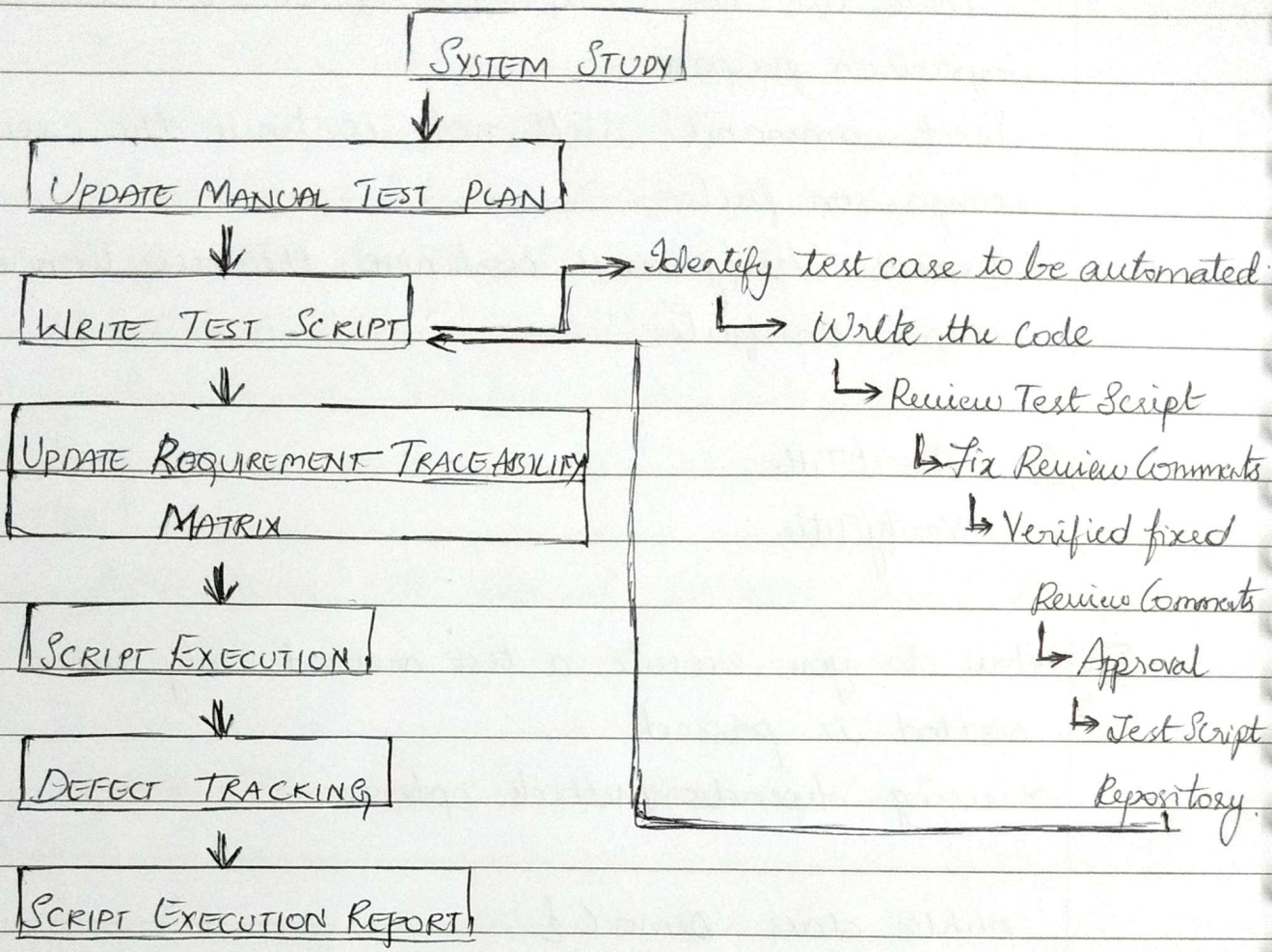
15/03/2017

Automation Test Life Cycle:



Automation test life cycle is a step by step procedure used/followed by automating the application. The first stage is system study, where we refer test case to understand the application.

In automation most of the cases, we do not create seperate test plan, instead we update test automation section of manual test plan itself.

In test automation, we specify the details of automation like the tool used for automation, framework used for automation, roles and responsibilities of each automation engineer etc.

Once the plan is ready, we start automating the application. While doing automation first we identify test cases to be automated.

Ex The test case should be part of regression Suite. This information will be provided by manual team. The regression test case should not have any manual intervention such as photo scanning, swiping of access card, Credit Card, Debit card, OTP, Captcha (completely automated public turing test to tell Computers and Humans apart), biometric scan, multimedia content etc., animations

→ Because of above reasons 100% automation may not be possible.

After identifying the test case, we write the code using automation framework. Then we send it for review & then we fix the review comments & again it is verified to check whether review comments are fixed or not.

After fixing review comment, it will be set to lead for approval and it will be stored in Test Script Repository after the approval.

Ex for repository is gitHub.

To ensure the coverage of automation ~~of the~~ we update requirement traceability matrix.

Req ID	TC ID	Automation	
R1	T1	YES	→ Automated
R2	T2	NO	→ Not yet automated
R3	T3	N/A	→ We are not automating.

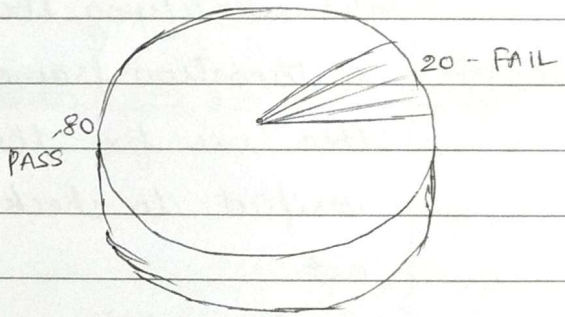
T3. cannot be automated because it may not be a regression test case or it might have manual interventions

When we get new build, we execute automation scripts, After the execution we analyse the result & if there are any failures because of defect in application, please submit the defect (defect tracking).

Then we prepare script execution report & publish it.

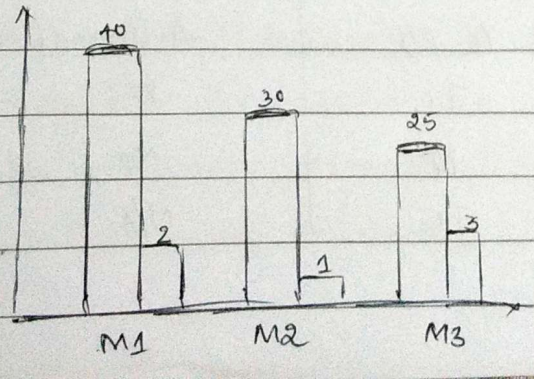
Detailed Report:

S.No	TC	Status
1	TC1	PASS
2	TC2	FAIL
⋮	⋮	⋮
100	TC100	



Summary Report:

Module Name	Pass Count	Fail Count
Module 1	40	2
Module 2	30	1
Module 3	25	3



__/__/____

AUTOMATION FRAMEWORK:

While developing automation script, all the engineers should follow same format to write the code to maintain consistency in the script. This will help us in maintaining the script & it will also help other engineers or new engineers to take over the job, when the person quits the job or not available for few days.

In automation this standard format is called as automation framework.

Defn: Automation framework is a standard rule, guideline and best practises which should be followed by all automation engineers while automating the application.

16-03-2017

Automation Framework Design: This is the first stage where automation lead/architect will specify files & folder structures and the generic code.

Ex: Step1: Create a folder in the required location (D:\BCSM15)

In the eclipse, go to File > Switch Workspace > other > Browse and select above created folder > Click OK.; which will restart the eclipse.

Step2: Create a java project (acti TIME)

Step3: Create following 3 packages inside "src"

- → Generic
- → pom
- → Script

Step4: Associate TestNG.

Step5: Create a folder with the name Drivers inside the java project & copy paste chromedriver.exe and geckodriver.exe.

Step6: Create another folder with the name jars inside the java project and copy paste Selenium server standalone jar file. Right click on the jar file > Go to Build path & select add to Build path.

Abstract class, Interface & Concrete class inheritance in the framework

1) In the automation framework, some of the values will never change, such as KEY & VALUE of drivers executables

2) To indicate that these values will never change, we make them as a constants and all the constants are generally stored in an interface.

Ex: Create an interface inside generic package as shown below.

```
public interface AutoConst {
    String CHROME_KEY = "webdriver.chrome.driver";
    String VALUE1 = "./drivers/chromedriver.exe";
    String GECKO_KEY = "webdriver.gecko.driver";
    String VALUE2 = "./driver/geckodriver.exe";
}
```

3) Every test case will have some common steps such as opening the appln & closing the appln. Instead of writing these codes multiple times, we make them as a method & we use inheritance along with annotations; so that these steps will be automatically

executed for every test. Inheritance is used for method reusability but we cannot directly execute the parent class, as there will not be any test method. To Indicate that parent class is incomplete without the child class (Test Method) we make the parent class as abstract.

Ex: Create a class with the name BaseTest inside generic package as shown below

```
public abstract class BaseTest implements AutoConst {
    public WebDriver driver;
@BeforeMethod
public void openApplication() {
    System.setProperty (GECKO-KEY, VALUE2);
    driver = new FirefoxDriver();
    driver.get ("http://www.google.com");
    driver.manage.timeouts.implicitlyWait (10, TimeUnits.SECONDS);
}
@AfterMethod
public void closeApplication(){
    driver.quit();
}
}
```

17-03-2017

Inheritance in POM: In selenium, for every page we create a POM class & in every page we perform some common actions. To avoid the repetition of the methods we use inheritance concept in POM class.

Ex

1) Create a class with the name BasePage inside generic package as shown below:-

```
package generic;

public abstract class Basepage {
    public WebDriver driver;
    public WebDriverWait wait;

public Basepage (WebDriver driver) {
    this.driver = driver;
    wait = new WebDriverWait (driver, 5);
PageFactory.initElements (driver, this);
}
public void verifyTitle (String eTitle) {
 try {
    wait.until (ExpectedConditions.titleIs (eTitle));
    Reporter.log ("Title is same", true);
 }
 catch (TimeoutException e) {
   Reporter.log ("Title is not same", true);
    Assert.fail();
   }
}
public void verifyElementIsPresent (WebElement element) {
 try {
    wait.until (ExpectedConditions.visibilityOf (element));
    Reporter.log ("Element is present", true);
}
catch (TimeoutException e) {
Reporter.log ("Element is not present", true));
Assert.fail(); }}}
```

FRAMEWORK IMPLEMENTATION:

Process of converting manual test cases into automation scripts using the designed framework is called as framework Implementation. In this process we automate only the regression test cases which do not have any manual intervention.

Ex: TestCase #1 : Valid login & logout.

→ Enter Valid username
→ Enter Valid password
→ Click on login button
→ Verify Enter Time Track page is displayed.
→ Click on Logout.
→ Verify that login page is displayed.

Ex: TestCase #2 : Invalid login

→ Enter invalid username.
→ Enter invalid password.
→ Click on login button
→ Verify error message is displayed.

Ex. Test case #3: Verify product version

→ Enter valid username
→ Enter valid password
→ Click on login button
→ Click on help (?)
→ Click on About actiTIME
→ Verify that product version is actiTIME 2016.1
→ Close the pop-up
→ Click Logout.

NOTE While converting test case into script, we develop two types of classes POM & test class. Senior engineers will develop POM class, junior engineers will develop test class.

Developing POM class:

Steps to develop POM class:

Step1: Execute the given test case manually which gives more clarity on the test case to be automated. While

Step2: executing the test case note down, the page elements and the actions.

Step3: For every page create a class inside POM package and the name of the pom class should be same as title of the page, ending with the word "Page".

~~Step~~:

NOTE All the POM class should extend BasePage class.

Step4: In POM class we should declare element using findBy annotation, initialize using PageFactory & utilize it using methods.

Ex: Page#1 · Login Page

Page elements un, pwd, login, errMsg.

Actions
1. Enter UN
2. Enter PWD
3. Click Login
4. Verify errMsg is present.
5. Verify title. (already present in BasePage).

__/__/____

Page #2 EnterTimeTrackPage

Page elements logout, Help, AboutactiTIME, productVer., CloseBtn

Actions
1. click logout
2. click help
3. click aboutAT
4. Verify Version
5. Click closeBtn
6. Verify Title.

LoginPage.java

When parent class has parameterized constructor, then it is mandate to use super statement in the child class/extending class.

Using explicit super statement: If parent class has parameterized constructor, then in child class it becomes mandatory to use super statement explicitly in user defined constructor.

```
Class A {
    A()        → default constructor
    :
}

class B extends A {
  B() {
    super();
  }
```

```
class A {
    A(int i)   → parameterized constructor
    :
}

class B extends A {
  B(int i) {
    super(i);
  }
}
```

loginPage.Java

```
package pom;

public class LoginPage extends BasePage {
    public LoginPage (WebDriver driver) {
        super(driver);
    }
}
```

18-03-2017

POM class for login page:

```
package pom;

public class LoginPage extends BasePage{
    @FindBy (id="username")
    private WebElement unTB;

    @FindBy (name = "pwd")
    private WebElement pwTB;

    @FindBy (xpath = "//div [. = 'login']")
    private WebElement loginBtn;

    @FindBy (xpath = "//span [contains (., 'invalid')]")
    private WebElement errMsg;
    public LoginPage (WebDriver driver) {
        super(driver);
    }
```

```
public void setUserName (String un){
    unTB.sendkeys(un);}

public void setPassword(String pw) {
    pwTB.sendkeys(pw);}

public void clickLogin(){
    loginBtn.click();
}

public void verifyErrMsgIsPresent(){
    verifyElementIsPresent(errMsg);}
}
```

Pom class for homepage:

```
package pom;
public class EnterTimeTrackPage extends BasePage {
  @FindBy(id = "logoutLink")
  private WebElement logoutLink;

  @FindBy(xpath = "(//div[@class='popup-menu-arrow'])[last()]")
  private WebElement help;

  @FindBy(linkText = "About actiTIME")
  private WebElement aboutActiTIME;

@FindBy(xpath = "//span[@class='productVersion']")
  private WebElement productVersion;
```

```
@FindBy(xpath = "//img[@title = 'close']")
    private WebElement closeBtn;

public EnterTimeTrackPage(WebDriver driver) {
    super(driver); }

public void clickLogout(){
    logoutLink.click(); }

public void clickHelp(){
    help.click(); }

public void clickAboutActiTIME(){
    aboutActiTIME.click(); }

public vord verifyProductVersion(String eVession){
    String aVersion = produtVession.getText();
  Assert.assertEquals(aVession, eVession); }

public void clickCloseButton(){
    closeBtn.click(); }
}
```

Developing Test Class:

1) No of test classes should be same as no of manual test cases
2) A test class should be created inside script package
3) Name of the class should be same as test case name.

4) Every test class should extend BaseTest Class
5) Inside every test class, we should develop test method using '@Test' annotation
6) Name of the test method should start with the word 'test'
7) Inside the test method write test case steps as inline comments; so that we will not skip any steps. and it will also help us while reviewing the script.
8) Below every comment call the required method of pom class as per the test case steps

Automation script for testcase 1:

```
package script;
public class ValidLoginLogout extends BaseTest {
 @Test
  Public void testValidLoginLogout () {
   //enter valid username
    LoginPage l= new LoginPage(driver);
   l. setUserName ("admin");
   //enter valid password
   l. setPassword ("manager");
   // click on login
   l. clicklogin();
   // verify Enter Time track is Displayed
    EnterTimeTrackPage e= new EnterTimeTrackPage (driver);
   e. verifyTitle ("actiTIME - Enter Time-Track");
   // click on logout
    e. click on logout ();
  // verify login Page is displayed
   l. verifyTitle ("actiTIME - Login"); } }
```

Assgn Test Case #2.

Test Case 3 → VerifyProductVersion.java

```
package Script;

public class VerifyProductVersion extends BaseTest {
  @Test
  public void testVerifyProductVersion(){
// enter valid username
    LoginPage l = new LoginPage(driver);
   l.setusername("admin");
   // enter valid password
     l.setpassword("manager");
   // click on login
     l.clicklogin();
    // click on Help
   EnterTimeTrackPage e = new EnterTimeTrackPage(driver);
    e.clickHelp();
    // click about ActiTIME
    e.clickAboutActiTIME();
    // verify that product version is 'actiTIME 2016.1'
     e.verifyProductVersion("actiTIME 2016.1");
    // close button click
     e.click_closeButton();
    // click logout
    e.clickLogout();
  }
}
```

TestCase #3

package script;

```
Public class InvalidLogin extends BaseTest {
  @Test
  Public void testInvalidLogin(){
     // enter invalid username
          LoginPage l= new LoginPage(driver);
          l.setUsername("abc");
     // entes invalid password
          l.setPassword ("damages");
     // click on login
          l.clickLogin();
     // verify Errmsg
          l.verifyErrmsgIspresent();
   }
}
```

MOCK INTERVIEW QUESTIONS:

11) What is TestNG? Why we use it & what is the execution order of @ (annotation)?

12) What is pom? Why do we use it?