TEST EXECUTION EFFORT ESTIMATION (TEEE) MODEL IN EXTREME PROGRAMMING

1MUHAMMAD DANYAL ASHRAF, 2PROF. NAEEM U. JANJUA
1MS in Software Engineering, SZABIST, Karachi, Pakistan
2Prof., Bahria University, Karachi, Pakistan
E-mail: 1danyal-ashraf@hotmail.com, 2njanjua@alumni.nottingham.ac.uk

ABSTRACT
Nowadays applications are being built by using Agile Methodologies. There are small iterations under each release. Customers write user stories and after prioritizing them are included in the iteration. The communication is very high with the customer. There is a continuous integration of code and builds take place to the customer very frequently due to which customer’s feedback comes quickly and rapidly so in this situation it is very important to have quality build release. For this it is vital that whatever the testing needs to be done in Agile software development is properly estimated. In this research paper, different approaches and techniques have been studied for Test Effort Estimation. After studying and comparing these approaches and techniques in detail, it was observed that there are still some drawbacks. To cover up these drawbacks, we have proposed the “Test Execution Effort Estimation” TEEE Model based on user stories which can be helpful for those applications which are developed by using Extreme Programming as an Agile Methodology. Low-level test estimation and high-level test estimation is also proposed through this model. And at last, we have also evaluated the proposed model through an empirical study.

Keywords: Agile Manifesto, Extreme Programming, Scrum, Test Estimation Models, techniques, Test Execution.

1. INTRODUCTION
Applications Agile software methodology is customer oriented and widely used now days in developing software in an organized and controlled manner. In Agile, we break the tasks into sub tasks with minimum planning and do not directly involve long term planning. Iterations are of three to four weeks. Each iteration goes through the complete cycle which includes planning, requirements analysis, design, coding and testing. It is not necessary that an iteration has sufficient functionality but the objective is to have build release with lesser number of defects.

There are different agile software development methodologies being used and each methodology holds different benefits and challenges. During my research, I have centered out the widely used Agile Development Methodology which is Extreme Programming. Extreme Programming can not be used independently, there must be some other thing needed to manage the project so in my independent study, I have also explained Scrum to manage and control project in Extreme Programming. In extreme programming, build for each iteration takes place very quickly to the customer for feedback so it is very important that whatever the testing needs to be done in Agile software development is properly estimated. This facilitates reducing the risks of project in terms of delays and cost because if we will have properly planned and estimated testing efforts then Project will be delivered on time without any time delays and ultimately we will have controlled cost of Project due to planned testing efforts.

Normally in SDLC, the effort estimation of overall project is given which covers from planning phase to maintenance phase. There are number of estimation models have been offered over years i.e. COCOMO which is used to estimate the effort of developing the software via lines of code, FPA is used for estimation via function points, Test Point Analysis is used to estimate the effort of all functional test activities which are based on use case points. Yet these models do not estimate the execution effort of user stories.

Aim of my independent study is to study the test effort estimation techniques/methods in Agile Software Development and what are the factors being considered before estimating the testing
effort. Further I have looked into manual test execution effort, test automation effort, test effort estimation based on execution of functional test cases; problems in the studied techniques/models and by considering those problems, I have proposed the solution to estimate the software testing execution effort in the better way to amend the quality of software being developed by using the particular extreme programming as an Agile Methodology.

2. RELATED WORK: TEST EFFORT ESTIMATION

During the last many years several models and methods were produced for effort estimation but most of them acquire effort estimation for the complete software rather than of software testing. The few models and techniques which I found and studied for testing effort estimation are:

2.1 Effort Estimation for Manual Test Execution

In this estimation model [1], the input is test suite and the output is effort estimation after executing the required test cases in the test suite. First we analyze each test step of the selected test cases and assign execution points (a unit of work to define the size and execution complexity of test cases). The execution point will be assigned to each step of every test case and then summing the execution points of each of its step will give you the execution complexity of a test case as shown in Figure 1. After assigning executions points for all the test cases, we will sum them to get the execution complexity of a complete test suite. On the basis of total number of execution points, we calculate the estimated effort by using test productivity or risk factors (similarly to COCOMO) [2].

2.2 Effort Estimation for Test Automation

Automated tests are cheaper to execute but hard to create which increases the cost. A number of automated tests can be managed by a tester at the same time and they can be scheduled to execute automatically at any time (usually in nights or in weekends). The effort estimation model is based on the test specifications written in controlled natural language (CNL) which is a subset of natural language that helps to write sentences in more précised and summarized way. This helps to identify the list of characteristics and risk elements that are related to estimate effort for test automation. The best automated test case is that which has higher manual execution effort and lower automation execution effort [1].

2.3 Test Execution Effort Analysis against Test Coverage

It’s always important to accomplish the highest level of test coverage while executing tests. For this, the tester has to prioritize the tests to execute them. Suppose there are 60 test cases so for test coverage we can prioritize them that 30 test cases are functional, 20 test cases are for input validation and 10 test cases are navigational. If a tester is experienced so it is feasible for him to distinguish the worth of every test case for the test coverage before executing. Using the above effort estimation model in 2.1, we can estimate the effort of every test case independently. For good test coverage, it’s important to select the best set of test cases which require high coverage and less execution effort [1].

2.4 Test Point Analysis

This method is required to estimate the effort of all functional test activities which are based on use case points. In Test Point Analysis, we calculate the estimated effort by using test productivity or risk factors (similarly to COCOMO) [2].

2.5 3-Dimensions Vector for Test Execution Effort Estimation

This approach was introduced to facilitate test execution effort estimation for regression tests. In this approach, test suite is characterized as a 3-dimensions vector which consists of three parameters: test case number, test execution complexity and tester itself (tester rank). Test case number is the total number of test cases in the test suite which are required for execution. Test case number can simply be taken from test suite. Each test case is different from each other because of their complexity so testing execution complexity is innovated. Test execution complexity is measured from execution effort point of view as mentioned in
2.1. Different testers rate different time for executing the same test case or test suite. For correct estimation it’s necessary to measure tester. Here in this model, the criteria to rank a tester in two different ways: how experience the tester is and how much the tester has awareness about the target application. On the basis of the above 3-dimensions vector model, a skilled database is set and at the end a sort of algorithm is used for effort estimation of a specified test suite vector [4].

2.6 Model-Based Estimation Approach in Testing Environment

In [5], an organized approach is mentioned for estimating the cost and time in testing environment. Firstly a comprehensive list of work items are documented w.r.t. features which are used for project estimation. Secondly the assumptions are made depending on the nature of each future which will be executed on a project (i.e. no. of test cases per feature). If assumptions are not made properly then it is not possible to make the estimated evaluation and there is a heavy chance of inconsistency in the estimation. The assumptions are made by keeping in mind the complexity and occurrence of each task. After that, the calibrated and referenced data is gathered from the previous projects. In data section, a brief description of each element is given which needs to be used in next step for producing model for each work item. This description is linked with one or few assumptions. And finally the value is assigned to each data description. Once the estimation system has a list of work items, assumptions, standardized collected data collection from the earlier projects, it’s simple to create model for each work item. The model will be the product of the expected number of iterations and the guessed effort for each iteration. Where possible, a model can be produced for each work item. These models should be evaluated by the management or subject matter expert (SME). Tasks which have some risk involved should be furthered reviewed when extra information is presented. The majority of the estimates of the work items are based on assumptions and the data documented. Any estimates can be revised by updating the assumptions.

2.7 Automated Test Execution Estimation Based on Functional Test Specification

In this model, a tool is designed that covers automated test execution effort estimation on the basis of the measurement of test size and test execution complexity. This tool is usually recommended for regression testing where we need existing test cases for execution. This tool has a mechanism to support natural language processing which recognizes the test action of every step of the test case which is based on the main verb of the sentence. Then the execution points are allocated to a test case or test suite that indicates how huge and complex it is. This is measured by evaluating the actions in the test cases w.r.t. the feature characteristics. There are number of models used for estimating effort of test execution on the basis of execution points and cost drivers. The cost drivers are factors connected with testing team and environment i.e. how much the team is experienced and what are the tools which support the environment. This automated tool allows different estimation models for use such as productivity-based estimation model and regression model. A productivity-based estimation model is a time per execution point (time / execution point) while doing test execution. Regression models look upon execution points and the outcome of the cost drivers for estimating effort of test execution [6].

3. MANUAL TESTING EFFORTS IN AGILE

3.1 Plan the Testing Effort

In [3], it is mentioned that how to plan the test effort in Agile. Mostly the testing schedule and its association with other project tasks are planned separately due to which the testers are unaware of it. So for this time boxes are given for each test levels. In time boxes, the plan splits into different time periods normally four to six weeks. This same approach is generally used to test each test level. The testing sub-team is advised to prepare a testing backlog in each time box that represents ‘test targets’ (a list of points that need to be tested). In other words we can say that backlog is a guide for testing sub teams to make their analysis and estimation for the upgraded version. In agile, the backlog is being used constantly to plan and guide the test effort.

3.2 Executing the Tests

Testing sub-teams run, inquire and explore each individual backlog entry until they meet its utilization that it should be consistent as compared to the previous versions of the system. It is very necessary to have professional decision that it is passed or failed. In case of failure, the tester must provide ‘steps to reproduce’ or screenshot or a reference document as evidence. The ‘test status’ column in the backlog must be updated to either passed or failed after running the test. Hence the ‘test status’ column should be planned, passed or failed [3].
3.3 Track the Testing Effort

To track the test effort in agile [3], burn-down chart is used within each time box. It is recommended that each backlog has its own burn-down chart so that the sub-team can easily interpret and analyze their testing progress. To set up the limit, an ideal burn-down line is drawn on a chart to give a sense of what to require. A compiled view of the testing progress for every time box is also required that would sum up the efforts from all sub-teams. To achieve this, we associate the data from all the worksheets into single backlog worksheet and then create burn-down chart of that single backlog worksheet as showing in Figure 2.

![Burn-down Chart](image)

Figure 2: A burn-down chart showing an ideal burn-down line and the actual test progress for every time box [3]

4. HYPOTHESIS

Hypothesis is basically a planned guess about the outcome of a problem or the status of the condition at the completion of the research. Here the hypothesis is:

\[ H : \text{To provide a test effort estimation approach for Agile Development}. \]

Here I have provided a "Test Execution Effort Estimation" approach/model for Extreme Programming as an Agile Methodology to confirm the hypothesis.

5. AGILE METHODOLOGIES

The word Agile means to move quickly and lightly. There are different agile software development methodologies being used i.e. Extreme Programming (XP), Open Unified Process (OpenUP), Feature Driven Development (FDD), Lean Software Development, Scrum, Dynamic Systems Development Method (DSDM) etc., so there are different benefits and challenges held by each methodology. In my independent study, I am focusing on Extreme Programming and Scrum methodologies.

5.1 Extreme Programming

Extreme Programming is the widely used agile methodology now days. The roles for the extreme programming are: XP Coach whose job is to introduce processes and explain it to the team and is also responsible for the team to be on the track and follow the processes. There is XP Customer in extreme programming whose role is to write user stories that hold business values and give it to the team. He is the one who decides that which tasks need to be included in the iteration. Once user stories are selected for the iteration, the customer and tester sit together and write acceptance tests for user stories in which the tester runs the tests and generating the results for a team. There is XP Programmer in extreme programmer whose role is to implement the code against user stories. XP Programmer is also responsible to perform Test Driven Development approach for implementing the user stories which is defined in the later paragraph. There is also XP Programmer – Administrator who is responsible to setup the environment of the programmer. XP Tracker is another role in extreme programming who is responsible to track iteration progress, release progress and acceptance test and communicates time by time to the management. Finally here comes XP Tester who is responsible for executing the tests, enter defects against the tests, make test summary report and send it to the whole team to have a look to it. [13]

In extreme programming [7], there is a concept of user stories written by customers. User stories are basically the replacement for large requirement documents. As the user stories are developed, release planning meeting takes place in which release schedule is created. Under each release, there are small iterations. For the beginning of each iteration, iteration planning meeting is required to develop the plan for each tasks. Open work space is recommended in extreme programming i.e. adding white boards for writing important notes etc. In extreme programming, we simply sum up the estimates of all user stories to measure the project velocity that were completed in the iteration. This measurement is useful in iteration planning; customers are restricted to select the same the number of user stories which has the same project velocity that was measured in the last iteration.
cards are used for designing the system. In XP, there is very less documentation; communication with customer is required in every phase, usually recommended face to face communication. All the code is written by two people with pair-programming concept in which 2 programmers working on a single computer. In XP, it is preferred that whatever the change in the code immediately be integrated so that all can work on the most upgraded version.

In XP [8], there is a concept of Test Driven Development (TDD). It is basically performed by XP Programmers. This strategy consists in five steps: 1. Make unit test cases from user stories; 2. Run the unit test cases and mark them fail because at this time code is not implemented; 3. Implement the code which let the unit tests to be passed; 4. Run again all the unit test cases to make sure that all the cases should be passed with the implemented code; 5. As the unit test cases passed; the developer starts refactoring the code to improve the consistency and performance of the build system. On some places, it is recommended that the cases should be automated unit test cases. Whatever the cases are, either its manual or automated, but TDD is the best approach to find errors at the start of the development. After the completion of iteration; user stories are delivered to the customer for the verification of user stories.

In Scrum, there is also meeting held on weekly basis named Weekly Scrum of Scrum; one member from every team is selected and takes part in the meeting. Through this meeting we can get the overall progress of the project. There is also Retrospective Meeting in Scrum which held after the completion of sprint to discuss the good and bad things about the last sprint and also discuss the improvements needed for the next sprint [10].

### 6. ESTIMATING TEST EFFORTS

#### 6.1 Who needs Test Effort Estimation?

Testing effort estimation is required by the management, customers and testing associations [11].

Management; here it is meant by the C.T.O., C.E.O. and managers from different teams. The whole management team requires effort estimation for the testing so they all ask to the testing manager or testing lead to provide them a testing estimation so that they can communicate accordingly with the customer.

Customers are who handed over their software development to a vendor. Their aim is to make sure that they should get what they have paid for.

Testing Associations: Now days there are testing associations mainly in European countries, their path of business is to check the quality (testing) of other’s software and also give them a verification certificate. So first thing after getting the requirements from the customer, they claim for the testing estimation.
6.2 Factors before doing Test Effort Estimation

There are some factors which require consideration before doing test effort estimation which are:

- System size
- Testing types required
- Testing tasks required
- Test reporting or non-testing activities
- Testing cycles
- Think of your past experience while making estimation.

6.3 What's the best approach for test estimation?

The best approach is particularly depends on the organization and the experience of the people implied in the project. For example, suppose there are two projects with same size and execution complexity. One project is related to any game and other project is critical medical treatment for patients. The test effort for critical medical treatment project might be very larger than the other project which is a game. So test effort estimation might be suitable for one project but may not be appropriate for another.

6.4 Issues in Estimating the Software Testing

The issues/problems due of which software test estimation is difficult are:

1. While test estimation, there is one confusion that which testing types and methods will be used during testing; mostly customers are unaware of the testing types and methods; the more testing types causes the increase in test effort estimation and also cost but customer always insist to reduce the cost so for reducing cost, testing types will be cut down which causes impact in the quality of the software [12].

2. The other problem which testing manager faces: one validation field can be tested with three input measures so is this one test case or three test cases? [11]

3. The problem which testing manager might faces during estimation is the test case coverage that what percentage of testing is covered by test cases and scenarios and what percentage of testing is covered by the experience or the natural knowledge of the tester.

4. Size of test cases varies at different stages of testing i.e. unit testing, integration testing, system testing etc. So there should be some normalization factor available to bring all the test cases to a standard size [11].

5. Now days the scope of automation is very high but there are not many resources available in the market for automation; if available but not much knowledgeable regarding automation so the test manager usually face problem that what exactly the effort will be required for automation.

6. Sometimes the manager or lead do estimates for the whole release so those estimates are mostly inaccurate. The best approach is to make estimation for each single iteration to avoid waste estimates [14].

7. PROPOSED TEST EFFORT ESTIMATION APPROACH IN AGILE DEVELOPMENT

After studying and comparing above models and techniques in detail, I have proposed “Test Execution Effort Estimation (TEEE) Model in Extreme Programming” which is based on user stories that can be helpful for those applications which are developed by using Extreme Programming as an Agile Methodology.

Here in my proposed approach, I have pointed user stories as test targets and on the basis of detailed and refined user stories, I have calculated “Test Execution Effort Estimation” approach. Refer the following diagram for more understanding.

![Diagram of Test Effort Estimation Model](Image)
7.1 Low-level Test Estimation

Based on previous iteration’s test execution estimation, we can estimate our current iteration’s testing execution through this proposed model.

\[ N - 1 \]

Here \( N \) is the current iteration and \( N-1 \) is your last iteration so you need to consider your last iteration’s estimation while estimating your current iteration.

In this model, I have created ‘metrics’ containing rules and also defined the scale with their standard weights. Here the scale is defined as Low/Medium/High with their weights 1/2/5. These are the standard weights based on your previous iterations estimations.

![Figure 5: Metrics indicating Rules, Scale, Weight and Time in seconds [self]](image)

After analyzing each user story in detail, firstly we will assign (execution points) to the respective characteristics. These execution points will be in seconds. Based on the application knowledge/user stories analysis, we will decide that under which scale the execution point will lie in. For more clarification, take an example (i.e. navigation between screens); as per the proposed metrics, if the navigation between screens for a particular user story is in between 3 to 5 then it will lie under medium scale which is weighted 2. Now assign the execution points for the respected characteristic i.e. 12 seconds. Execution points will mostly be different depending on the complexity of the user story. For getting the complexness level, we multiply execution points by weight which is in this case is 12 * 2 = 24 seconds is a complexness level of a user story in terms of navigation between screens. Now calculate the complexness level of each user story and sum them all and then divide it by 3600 to make the estimation in man-hours.

![Table 1: Complete Metrics for Test Execution Effort Estimation [self]](image)

The complete metrics containing rules, scale along with their weights and the execution points is as follows in which the execution points are assigned to those characteristics that are accomplished by a particular single user story.

![Table 2: Showing Estimation in seconds by multiplying EP with Weight [self]](image)
The above table shows the estimation of 10 user stories which are derived from the proposed metrics. Divide it with 3600 to make estimation in man-hours.

\[ \frac{7264}{3600} = 2.01 \text{ man-hours} \] for testing 10 user stories.

### 7.2 Defining Weights

For the current iteration estimation, the weights of the previous iteration’s estimation are defined. These weights may vary for the iteration. For calculating weights, first group the user stories w.r.t. the low complexity level, medium complexity level and high complexity level. In the current iteration (i.e. for low complexity level, the estimated time was 10 hours and after completing testing, the actual time is 15 hours).

**Actual Time - Estimated Time**

15 - 10 = 5 hours

The difference is the 50%. Now perform root cause analysis that why this difference occurs. After the root cause analysis, if it proves that the wrong estimation was made then this 50% will be added to the weight of low complexity level which is actually 1 and after adding it will make it 1.5 as it is appearing in the below table. Then this changed weight will be used in the next iteration’s estimation.

### 7.3 High-level Test Estimation

I have also proposed a high-level test estimation by dividing the estimation with execution points.

<table>
<thead>
<tr>
<th>User Stories</th>
<th>Execution Points</th>
<th>Estimation (sec) EP * Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>350</td>
<td>700</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>650</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>600</td>
</tr>
<tr>
<td>4</td>
<td>425</td>
<td>925</td>
</tr>
<tr>
<td>5</td>
<td>570</td>
<td>1425</td>
</tr>
<tr>
<td>6</td>
<td>170</td>
<td>210</td>
</tr>
<tr>
<td>7</td>
<td>153</td>
<td>288</td>
</tr>
<tr>
<td>8</td>
<td>410</td>
<td>328</td>
</tr>
<tr>
<td>9</td>
<td>340</td>
<td>620</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>180</td>
</tr>
</tbody>
</table>

**Total**

3905

7264

### 7.4 Why test execution estimation?

As we all know that testing software is an organized activity which usually holds test requirements (i.e. use cases document), test planning (i.e. test plan document), test design (i.e. test case preparation, test data and test script preparation), test execution (i.e. executing test cases and test scripts) and test reporting (i.e. test summary report, build release report etc.). Test requirements can be prepared in corresponding with business requirements; test planning and test design can be prepared in corresponding with development planning, designing and coding. Test reporting can be prepared in the defect fixing stage or delivering phase. Only test execution can not be managed in corresponding with other development activities and it is a critical part in the software life cycle. Estimating test execution effort can helpful to control the timeline of the entire project.
7.5 Advantages of the Proposed Test Execution Effort Estimation (TEEE) Model in Extreme Programming

1. There is no restriction for using any language for user stories i.e. CNL or natural language like in other test estimation models. You can either use CNL or natural language for writing user stories.

2. Through this model, test estimation is done at the start in the release planning after the creation of user stories and before the creation of test specifications and acceptance tests.

3. In the studied models, customization of the characteristics is required when doing estimation for the other company or any other project. In the proposed test estimation model, general characteristics are used to avoid customization when doing estimation for another project or any other company.

4. In the studied models, no one has mentioned the proper way to estimate the effort in terms of time but the proposed model is calculating the time estimation in a proper manner after calculating the execution points.

5. Proposed test estimation model is focused on the execution points of user stories rather than the execution points of every step of the test case, this avoid increasing the cost and time.

8. EMPIRICAL STUDY

I have applied the proposed model in one project of my organization which is currently being developed by using Extreme Programming and I have also attached the results regarding it. How I came to know that project is based on extreme programming. As in this project there is a concept of user stories, iterations, some sort of pair programming and more importantly, the development team is using test driven development strategy. First of all, one team member transfer me the knowledge about the project after that I requested the project manager to provide me the detailed user stories for the current iteration. There were total 55 user stories for that iteration. I sat along with the project manager and start estimating the execution of the user stories through the proposed model. I assumed the same weights for defining the complexity level. The total time which we took for estimating the execution of user stories was 2.5 hours.

The test execution estimation (EP * Weight) of all user stories was 287478 seconds which is 80 man-hours means 10 days.

<table>
<thead>
<tr>
<th>User Stories</th>
<th>Execution Points</th>
<th>Estimation (sec) EP * Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1701</td>
<td>3074</td>
</tr>
<tr>
<td>2</td>
<td>957</td>
<td>2744</td>
</tr>
<tr>
<td>3</td>
<td>1268</td>
<td>5741</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>55</td>
<td>94708</td>
<td>287478</td>
</tr>
</tbody>
</table>

Table 4: Showing test execution estimation results derived on other project [self]

9. CONCLUSION AND FUTURE WORK

In this research paper, I have discussed different models and techniques for test effort estimation and also the observed drawbacks and improvements of these models have been mentioned in this paper. Plan the testing effort, executing the tests and track the testing effort in agile is also being a part of this paper. This independent study tells that what research methodology has been used for covering the research. In this research paper, it is also been discussed that who needs test effort estimation, what are the factors while estimating the test effort, what’s the best approach for test estimation and what are the issues in estimating the software testing. As the proposed model is for extreme programming as an Agile Methodology so I have discussed extreme programming in detail and also discussed Scrum as an Agile Methodology which is used to manage and control the software in extreme programming. So after studying extreme programming and discussing different test effort estimation models/techniques and their drawbacks, a model for “Test Execution Effort Estimation (TEEE) in Extreme Programming” has been suggested based on user stories which can be helpful for those applications which are developed by using Extreme Programming as an Agile Methodology. After that I have done an empirical study to test the proposed model/approach. I applied this model in one of the project in my organization which is currently being developed by using Extreme Programming. There were total 55 user stories for that iteration. After applying the proposed model, the total time which the model calculated for estimating the execution of user stories was 80 man-hours. Even tough this model is applied for a limited data but after comparing with
the actual time; the results look fair so by taking this, we say that our hypothesis is Accepted.

In the proposed model, the factor of people (tester) can be considered in the future. Generally for executing the same tests, every tester takes different time so we can measure the tester as per knowledge on the application or as per experience which can be helpful for estimating the effort execution more accurately. The proposed estimated model does not provide guidelines about the risk factors or uncertainty acquired by the system. Including these factors in the guideline will extensively improve the quality of the estimation. Nevertheless estimates are just estimates and there is constantly an ambiguity/doubt connected with an estimate. So the best practice is to update the test estimates as the project knowledge grows time by time. This can make you sure that the promising estimates are obtainable.

REFERENCES


AUTOR PROFILES

**I Muhammad Danyal Ashraf** received the degree in Software Engineering from University of Huddersfield. Currently I am a research student and doing MS in SE from SZABIST, Karachi Campus. I have three years of experience in software quality assurance. My interests are software automation and software process improvement.

**I Prof. Naeem ul Hassan Janjua** received the degree in B.E. Electrical from NED University of Engineering and Technology and then did MSc Electronics from University of Nottingham. I was lecturer at NED, then Deputy Director at Hamdard University and currently Professor at Bahria University, Karachi Campus.