This whitepaper aims to outline a standard regression methodology that can be adopted for a testing project. It creates and interlinks standard set of parameters and processes, which produce quantifiable values for each test case. It talks about the extended traceability, challenges, overlooked parameters and their significance, process derivatives and finally tries to arrive at a standard algorithm for each test case. I have derived the thoughts from my project experiences in ‘data migration project’ and ‘SOA based - End to end business process testing project’ in Agile Delivery Model.

Author: Debabrata Panda (Debabrata_panda@infosys.com)

www.infosys.com

Powered by Intellect, Driven by Values
Executive Summary

Background

In the current project (Product set 1) there are over 70 business processes and almost 200 functionalities. Having run for over 4 yrs in Agile mode, the project has delivered the initial build of the entire Application and subsequently 250 change requests and almost 1000 service requests.

Each Change/Service request typically either adds to the business processes (horizontal add on) or brings in new functionalities (Vertical add on) or enhancement into the already existing functionalities/business processes. The Change requests are delivered in the form of user stories and acceptance criteria.

This has led to a cumulative figure of over 30,000 end to end test cases. Each test case on an average tests 10 systems typically the Order entry gateways, Integration layer, CRM applications, Orchestration & Mainframe applications.

Overall there are 5 product sets in the program with interlinked functionalities and processes. Looking at the magnitude of task at hand, it is extremely difficult to identify TCs impacted by CRs and TCs impacted by defects. This has a cascading effect on identifying the actual regression and Automation requirement, also the redundant TCs keeps on increasing without proper methodology and process guide.

This whitepaper proposes three methodologies to address the issue discussed and describes the process of its usage and sequence.

1. Regression pool creation using mnemonic code
2. Mechanism to identify the best TCs for regression
3. Regression Test case identification based on Defects Area of influence

Benefits accrued

The regression pack size was reduced by over 70%.

Total test coverage was increased by 38%.

Effort saved due to reduction in redundant work stands at 23%.

This helped in controlling the spiraling number of test cases and paved way for test parameterization, which helped in higher Test case design productivity.

Target Audience

Test lead
Test Analyst
Test Managers

Target Projects

Maintenance Projects

Large scale projects running in agile delivery model

Automation projects intending to identify best possible legacy test cases

Note: The project is a CMMi level 5 project and has successfully implemented six-sigma methodologies to significantly drive down the non-engineering effort and improve productivity by innovative methodologies.
1. Regression pool creation using mnemonic code

A regression pool is a test set created after synchronization process to eliminate redundant or duplicate test cases. The regression pool, once created, undergoes minimum changes irrespective of the type of changes brought in by the delivery. As and when new releases are delivered, a small part of the regression pool is replaced by the new functionalities.

Regression pool is a subset of all the functional test cases after normalization. It is different than the regression pack – Basically a regression pack is a summation of subset of regression pool + Subset of new functionalities.

Benefits

1. The regression pool covers all the functionalities and serves as a fixed target of automation and helps in avoiding automation effort wastages, which is a menace in the agile delivery model.

2. It is easier to operate on a standard small set of test cases rather than work on the entire set of functional test cases.

3. Most of the duplicate and redundant test cases are removed, which enables the tester to save his effort in NOT retesting, the ‘already tested’ functionality.

4. Finally, it helps in Risk based testing by eliminating the time constraint to execute very high number of test cases.

Mnemonic method is used to arrive at the regression pool from the exhaustive set of functional test cases.

**Key features of Mnemonic Method**

1. Considers all the functional test cases to segregate into static regression pool and dynamic regression pool.

2. Identifies duplicate and redundant test cases and removes them from the regression pool.

3. Creates parent-child relationship among TCs and ensures child TCs are not tested, if a parent is already planned to be executed.

4. Test case realignment across products

**Illustration from the Project**

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total TCs for Release 10</td>
<td>5600</td>
</tr>
<tr>
<td>Total TCs retained in Regression pool after filtration</td>
<td>1650</td>
</tr>
</tbody>
</table>
Process of arriving at mnemonics

i) Each Test case is mapped to a business process and Functional Scenario as given below

<table>
<thead>
<tr>
<th>Business Processes</th>
<th>Functional Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B Functionality 1</td>
</tr>
<tr>
<td>2 Business Process 1</td>
<td>TC_1</td>
</tr>
<tr>
<td>3 Business Process 2</td>
<td>TC_3</td>
</tr>
<tr>
<td>4 Business Process 3</td>
<td>TC_5</td>
</tr>
</tbody>
</table>

ii) The TCs are given mnemonic codes based on their presence in the traceability matrix.

Once the mnemonic codes are attached to the test case, it becomes easier to identify the test cases addressing similar functionalities and reduce the redundancy by sorting it out.

<table>
<thead>
<tr>
<th>Test case Name</th>
<th>Test case functionality capture</th>
<th>Mnemonic</th>
<th>Candidate for regression Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_1</td>
<td>Bus Pro 1 (Func 1 + Func 2)</td>
<td>2BC</td>
<td>Y</td>
</tr>
<tr>
<td>TC_2</td>
<td>Bus Pro 1 (Func 1 + Func 3)</td>
<td>2BD</td>
<td>Y</td>
</tr>
<tr>
<td>TC_3</td>
<td>Bus Pro 2 (Func 1 + Func 2 + Func 3)</td>
<td>3BCD</td>
<td>Y</td>
</tr>
<tr>
<td>TC_4</td>
<td>Bus Pro 2 (Func 1)</td>
<td>3B</td>
<td>N</td>
</tr>
<tr>
<td>TC_5</td>
<td>Bus Pro 3 (Func 1 + Func 2)</td>
<td>4BC</td>
<td>N</td>
</tr>
<tr>
<td>TC_6</td>
<td>Bus Pro 3 (Func 1 + Func 2 + Func 3)</td>
<td>4BCD</td>
<td>Y</td>
</tr>
<tr>
<td>TC_7</td>
<td>Bus Pro 3 (Func 1 + Func 2 + Func 3)</td>
<td>4BD</td>
<td>N</td>
</tr>
</tbody>
</table>

iii) Sample TCs are shown in the matrix, where the actual business processes and functionalities have been replaced with placeholders and the TCs name replaced with their "Quality centre Test Id". The table below shows, some sample test case mnemonics and the test cases replaced by it.

<table>
<thead>
<tr>
<th>Some sample Test case mnemonics from the existing project</th>
<th>TCS replaced</th>
<th>TCS replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>26BF'AC&quot;AN'</td>
<td>3</td>
<td>26B; 26FAC'; 26AN'</td>
</tr>
<tr>
<td>2B'AT'BF'</td>
<td>2</td>
<td>2B; 2AT'</td>
</tr>
<tr>
<td>2B'AO&quot;BF&quot;</td>
<td>3</td>
<td>2B; 2'A'O; 2B'BF&quot;</td>
</tr>
<tr>
<td>------------------</td>
<td>---</td>
<td>-------------------</td>
</tr>
<tr>
<td>2B'AL&quot;AZ&quot;BF&quot;</td>
<td>4</td>
<td>2B'AL&quot;AZ&quot;</td>
</tr>
</tbody>
</table>
2. Mechanism to identify the best TCs for regression

The rigid timelines in an agile mode effectively means that an E2E test planner has to manually choose the best possible combination out of a host of possible combination, instead of executing all the available scenarios. This mechanism helps creating the regression pack while unleashing ‘Risk Based Testing’ to the fullest without compromising the product quality.

Thus the regression pool identified through the earlier process has to undergo further trimming to find out the best possible scenarios that would ensure maximum coverage in the regression pack.

- Unlike usual test cases where an “input” necessarily gives an “output”, E2E business tests are different. Basically, an E2E test case is of the following structure:

  \[\text{Input} \rightarrow \text{Intermediate Outcome/Action} \rightarrow \text{Action based on outcome (OA1)…. } \rightarrow \text{OAn} \rightarrow \text{Output}\]

- In the E2E test cases most of the Functionalities are dependent – where, a string of preceding functionality determines the eventual outcome (functionality). Once the string is changed the outcome differs.

Proof of Concept from the project:

- The idea was to identify all the test cases using the mechanism discussed.

- The output will be compared to the testcases using the manual coverage techniques for the test coverage and the defects identified/slipped to downstream testing.

The existing test cases were broken into: Common factors and unique factors. The unique factors were put in a grid as shown below and the mechanism was applied.

<table>
<thead>
<tr>
<th>Type of order</th>
<th>Data Type</th>
<th>Look up table</th>
<th>Delay Code</th>
<th>Exception</th>
<th>Reaction</th>
<th>Amend</th>
<th>Notification Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process 1</td>
<td>Single</td>
<td>Opted for the Standard process</td>
<td>DC1</td>
<td>Call Desk Notification</td>
<td>Abort the order</td>
<td>Amend the date</td>
<td>Yes</td>
</tr>
<tr>
<td>Business Process 2</td>
<td>Double</td>
<td>Opted for the Enhanced process</td>
<td>DC2</td>
<td>Back End Notification</td>
<td>Amend the order</td>
<td>Amend the address</td>
<td></td>
</tr>
<tr>
<td>Business Process 3</td>
<td>Multiple</td>
<td>Opted for the Premium process</td>
<td>DC3</td>
<td>Engineer Notification</td>
<td>Manually progress the order</td>
<td>Amend the equipment</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We have noticed that in the manual approach,

- 23% of the Test cases were redundant.
- 11% Test cases were not captured
In addition to it, the above mentioned approach was able to reduce the regression pack to almost 55% without impacting the test effectiveness or the test coverage.

**Illustration and principles**

1. *Eliminate dependent functions* - Find out the dependent parameters and consider them as one. Ex. Input Action 2 determines Outcome 1.

2. *Group all possible combination* based on Action-outcome category – in the above case Input Action2 and Outcome 1 can be grouped together to form one parameter.

<table>
<thead>
<tr>
<th>Type of order</th>
<th>Data Type</th>
<th>Look up table &amp; Exception</th>
<th>Delay Code</th>
<th>Reaction</th>
<th>Amend</th>
<th>Notification Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process 1</td>
<td>Single</td>
<td>Opted for the Standard, Call Desk Notification</td>
<td>DC1</td>
<td>Abort the order</td>
<td>Amend the date</td>
<td>Yes</td>
</tr>
<tr>
<td>Business Process 2</td>
<td>Double</td>
<td>Opted for the Enhanced process, Back End Notification</td>
<td>DC2</td>
<td>Amend the order</td>
<td>Amend the address</td>
<td></td>
</tr>
<tr>
<td>Business Process 3</td>
<td>Multiple</td>
<td>Opted for the Premium process, Engineer Notification</td>
<td>DC3</td>
<td>Manually progress the order</td>
<td>Amend the equipment</td>
<td></td>
</tr>
</tbody>
</table>

Input

<table>
<thead>
<tr>
<th>InputAction 1</th>
<th>InputAction 3</th>
<th>OutputAction 1</th>
<th>OutputAction 2</th>
<th>Output</th>
</tr>
</thead>
</table>

3. Identify the *least dominant action* and reduce the variation – For example the selection of any delay codes has least impact on the OutputAction1 or OutputAction2. So, ignore the delay code.

4. *Similar scenario classification* – If there are a lot of business processes, try to find out similar business processes and group them under a parent process. Apply Orthogonality approach to the parent business process rather than the Child processes. For example, if Bus Pro 1 is similar to Bus Pro 2, replace them with Bus Pro1&2 and apply the orthogonality.
5. **Class grouping** – Once all the unique parameters are identified, combine them to form all the possible combination. In the above case, Output Action 2 is dependent on selection of “Amend the order” in Output Action2. Hence Output Action2 is not considered under unique combination category and will be treated for TCs specific to “Amend” during the consolidation phase.

<table>
<thead>
<tr>
<th>Type of order</th>
<th>Data Type</th>
<th>Look up table &amp; Exception</th>
<th>Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process 1&amp;2</td>
<td>Single</td>
<td>Opted for the Standard, Call Desk Notification</td>
<td>Abort the order</td>
</tr>
<tr>
<td>Business Process 3</td>
<td>Double</td>
<td>Opted for the Enhanced process, Back End Notification</td>
<td>Amend the order</td>
</tr>
<tr>
<td>Multiple</td>
<td>Multiple</td>
<td>Opted for the Premium process, Engineer Notification</td>
<td>Manually progress the order</td>
</tr>
<tr>
<td>Input</td>
<td>InputAction1</td>
<td>Param1</td>
<td>OutputAction1</td>
</tr>
</tbody>
</table>

6. **Class elimination** using pair wise elimination approach – Consider various possible combination of InputAction1, Param1 and OutputAction1. Give weightage to the functional criteria which controls the TCs like Input and Param and reduce the variation of OutputAction. Basically the parameter at the beginning of a scenario has more weightage rather than the parameter towards the end of TCs as illustrated in the graph below.

7. **Consolidation**

Once the unique set of TCs is identified add the least valued parameters (delay code in this case) and the dependent parameter based on the suitability.
3. **Regression Test case identification based on Defects Area of influence**

During a progression test execution cycle defects are identified and linked to the Test cases. These defects define the regression suite. We will discuss the various parameters and how they influence the selection of regression pack to be executed for a given release.

*This strategy has been successfully implemented in the last Major release.*

- **Number of defects linked to a Test case**

A test case can have traceability to many functional requirements and a functional requirement can be addressed by more than one test case. This makes for an interesting observation where a single test case can be linked to more than one defect and a single defect can be linked to more than one test case. In such scenarios it becomes imperative to identify the test cases with highest severity and very high probability of failure.

Two new factors were arrived from the Defect analysis report, “Likelihood of fix fail”, “Relative LoF”. They are defined as:

i.  
\[
\text{Likelihood of failure of a Test case} = \sum \left( \frac{1}{D_1} + \frac{1}{D_2} + \cdots + \frac{1}{D_n} \right)
\]

Where

\[
D_n = \text{No. of TCs associated with defect } 'n'.
\]

For each unique defect associated the value of 1/D will be equal to 1, whereas for a common defect associated with 4 TCs, the 1/D value would be 0.25.

ii.  
\[
\text{Severity of failure of a Test case} = \sum \left( \frac{DS_1}{D_1} + \frac{DS_2}{D_2} + \cdots + \frac{DS_n}{D_n} \right)
\]

Where

\[
DS_n = \text{Severity of the defect associated with the test case}
\]

Highest severity should have the highest computational value (e.g. High Severity should be of value 3 and low severity should be of value 1)

\[
\text{Test case composite score} = \text{Probability of failure of a Test case} \times \text{Severity of failure of a Test case}
\]

- **Area of Influence**

Once the test cases are sorted out based on the composite score, they are located on the mnemonic grid. Here we discuss about two areas of influence: Child Inheritance and Parent order tracking.

i.  
**Child Inheritance**

Say, test case [26BF'AC"AN'] is associated with a defect, all the test cases with the same combination and additional functionalities would fail. Let’s assume there are 5 such test cases.

ii.  
**Parent tracking**
For the same test case, verify the 1\textsuperscript{st} order of parent [26BF'AC'] and test it. If it fails, go to the 2\textsuperscript{nd} order of parent [26BF] and test for its failure.

**Normalization**

The highest order of parent gives the area of influence and all the child test cases are covered by it. So, if [26BF'AC'] fails but [26BF] is executed without a failure, the area of influence is ['AC'].

If the 1\textsuperscript{st} order of parent is tested successfully without a failure it would mean the area of influence is the test case associated with that defect. So, if [26BF'AC'] is executed successfully than the area of influence is ['AN'].

**Conclusion**

In an Agile environment where the changes are fast and time is crunched, Risk based testing is the only option available. The methods described above helps the team to flexibly fit in the testing window without impacting the delivery schedule or the test coverage.

The methods discussed are cyclic in nature which would mean they can be used in any order to reduce the regression pack based on the schedule and resources at disposal.

Though they have been discussed under the regression analysis phase, these methods can be used during the progression phase and the automation test case pack creation as well. The static regression pool gives an option for a steady automation without many changes in the “Business process reusable” irrespective of the functionality changes brought in by the delivery. These methods are not limited to the “SOA based end to end business process testing” but can also be generalized to include the variety of testing performed these days including web applications and SAP.

Through minor customization, this methodology can be used ‘as-is’ in all kind of testing projects cutting across domain, technology and delivery model. This has also significant impact in reducing the automation cost loss by automating test cases which are valid for longer duration. This methodology will be put into maximum use in Maintenance projects, where the chances of test cases duplication and effort redundancy are highest.

The mnemonic method described below has also been utilized for skill set management, transition and Knowledge management, and Effort estimation for overlapping functionality.
Reference

The ideas were developed and formulated during project progression stage and has no dependency or similarity to any other sources.

About the author

The author is a Test Lead in Infosys, with 5 years of experience in software testing and agile delivery model. He has specialization in Operations Support Systems (OSS), broadband provisioning and fault management - E2E Business process testing. He is presently working with a leading Telecom provider of Europe, as a Technical Lead - Testing.

Key Achievements

Apart from the delivery excellence, the author has also focused on the various process and quality initiatives.

- Built a smarter ‘test estimation model’ aligned with the complex delivery structure resulting in an effort deviation of 1.3% only. Was awarded Innovals’2008 award – Infosys Validation Unit recognition in the field of Innovations.
- Created a dynamic process for maintaining the run time skills of the team members within the OSS products and Business processes. Using the process the skills were quantified and were processed using mathematical tools to help in identifying the Knowledge management requirement, transition, personnel movement across products etc, as per the guidelines of a TCoE model.
- Created the ‘automated test planning tool’ resulting in a 40% test planning savings over traditional approach.

Academic Qualification

Bachelor of Engineering in Electronics and Communications - 2004,
Kalinga Institute of Technology and Science, Bhubaneswar (www.kiit.ac.in)

Certifications Awards

- Certified professional from QAI (Quality Assurance Institute) as a CSTE (Certified Software Test Engineer)
- PMElite Lite – Infosys Certification on Project Management
- Infosys internal certification in OSS-BSS, Telecom & datacom and OOAD.
- Innovals’2008 for Best practices in Estimation.
- Most Valuable Player of the quarter (Q3’ 07-08)
- IVS best project award in 2006-07.