Implementation of Effective Test Process Improvement and Test Execution to minimize the Cost of “Slippages”, “Defect Rejection” related to Client Deliverables


Capgemini India Private Limited
Crescent 2, Prestige Shantiniketan
Sardaramangala Village, Whitefield
Road, Bangalore -560048, India

Author: Sharad Kumar & Amrik Singh
Date: 09/16/2013
Email:

sharad.a.kumar@capgemini.com;
amrik.b.singh@capgemini.com;
Table of Contents

1. Abstract ................................................................................................................. 3

2. What is Slippage or Defect leakage and Defect Rejection? .......................... 3
   2.1 Slippage ........................................................................................................... 3
   2.2 Defect Rejection and why it occur ................................................................. 4
   2.3 Why do slippages or defect leakage occurs in spite of testing? .............. 5
   2.4 What are the main reasons behind this? ...................................................... 5
   2.5 Why severity and priority are the most important aspects for defects and its hidden perceptive .......................................................... 6

3. How to implement effective test process improvement and test execution tracking to minimize the cost of slippages, defect rejection .................. 7
   3.1 Static Testing of Business Requirements and Rules ................................. 7
   3.2 How to perform Static testing? ..................................................................... 7
   3.3 Static Test Plan ................................................................................................ 8
   3.4 How to write effective test cases using parameterization and modularity concepts in Quality Center (Test Plan) ............................................. 9
   3.5 How to perform test case execution with readiness for functional requirements with effective tracking? ......................................................... 13

4. Conclusion ............................................................................................................. 16

5. References ............................................................................................................. 17

6. Author’s Biography ............................................................................................ 17
1. Abstract

In current circumstances, software systems continue to grow gradually in density & extent. Business load for shorter development cycles have strained software development society to resist to discover among functionality with quality and time. In general need of proficiency, scheduled and unknown pressure, inadequate resources. These problems contain imperfect design, unproductive testing, deprived quality, maintenance cost and poor client satisfaction. Mode to prevent defect from being delivered or “leakage, slippages, rejection” to customers, it is very important to have a way to evaluate the current testing process for its potency & weakness and to highlight the risks & exposures that exist. As we all know that it will cost less to find the defect in early stage of process & certainly too much more expensive to fix them once they are on the ground. It is vital that all possible defects need to pick over from the software before it reaches to the end custom

The purpose of this whitepaper majorly focus to implement practical & effective test process improvement using test management tool (QC) to minimize the cost for slippages & rejection. Why do slippages or defect leakage occurs in spite of testing? What is defect rejection and why it occurs? How to calculate defect leakage and rejection? What are the main reasons behind this? Why severity and priority are the most important aspects for defects and its hidden perceptive. In this our focus area will emphasize the solution’s provided for challenges faced in each testing while performing STLC procedures (including JAD, what is static testing? how to perform it? how to write effective test case prep using parameterization and modularity concept in QC and how to perform test case execution with readiness for functional requirements with effective tracking & analysing the defect through defect life cycle).Key points to remember, advantages & disadvantages and lastly conclusion.

2. What is Slippage or Defect leakage and Defect Rejection?

**Slippage:**-Any defect or failure which is found post testing phase in the customer environments needs to be categorize as “Slippages”. There are 2 main phases which occur to be the key sources for the slippages are “Design phase and testing phase”. Design phase would source the most expensive defects or slippages and is mainly caused by requirements gaps e.g. unidentified requirements, inadequate requirement that results in error of slip by the program designer. An ordinary source of requirement gap is non-functional requirements such as testability, scalability, maintainability, usability, performance and security. Test phase is the stage where verification and validation is carried out for the software and this is area which will be analysed in upcoming sections.

Slippets are those for e.g. Suppose 10 business requirements (BUSR) can have 101 functional test cases to write and execute in ST phase out of which 90 are positive TC and 11 are negative TC. Testers’ somehow missed to cover 1 negative functional test case and covered 100 TC out of which 90 positive TC and 10 negative TC. Now, in production environment, unfortunate, that one missed negative TC impacted and High severity (S1) & High priority (P1) of defect occurred; here it will be treated as “Slippage”. Let’s explain phase wise defect slippages, for e.g. “there is some functionality implemented in system testing (ST) where few defects occurred, fixed, retested and closed following proper defect life cycle. Some interface defect found, fixed, retested and closed in system integration testing (SIT) following proper defect life cycle. Suppose Defect id for functionality in ST phase is 1 and Defect id for interface in SIT phase is 2.

Now in UAT, we found 1 defect which functionality is same as in ST for what defect id 1 was closed successfully and other defect found which interface is same as in SIT for what defect id 2 was closed successfully. Here it comes as functional and interface slippages in UAT. It might be possible that testers didn’t test the defect properly at the time of retest with particular time frame but it is most of the unfortunate situation in later test phase because of the improper standards of merged code deployed,
environment related issues, data corrupted and at insignificant amount of time: deployed code didn’t work properly. If the tester didn’t retest the defect properly in current test phase and closed it. Then in later stage, if that closed defect occurs, it would be treated as slippages. Defect leakage can be categorized in defect type as “functional, environment, data, and interface” etc.

How to calculate Defect leakage in percentage phase wise: - Pre-requisite’s should be at least defect was fixed and closed to calculate the defect leakage according to testing phase wise

1) If it is functional defect and occurred in SIT phase then formula would be

\[ \% \text{ Defect leakage to SIT} = \frac{\text{Defect slink from ST}}{\text{Total no of defect count (ST+SIT)}} * 100 \]

2) If it is functional defect and occurred in UAT phase, then formula would be

\[ \% \text{ Defect leakage to UAT} = \frac{\text{Defect slink from ST}}{\text{Total no of defect count (ST+SIT+UAT)}} * 100 \]

3) If it is an Interface defect and found in UAT phase, then formula would be

\[ \% \text{ Defect leakage to UAT} = \frac{\text{Defect slink from SIT}}{\text{Total no of defect count (ST+SIT+UAT)}} * 100 \]

How to calculate overall defect leakage in percentage:-

1) If no of functional/interface defect found in any phase, overall defect leakage In %=

\[ \% = \frac{\text{Total no of defect reported in production by client}}{\text{Total no of valid defect count during testing phase}} * 100 \]

**Defect Rejection and why it occur?** :-Defect rejection is basically treated as “Invalid Defects”. It occurs because of providing “improper information, inadequate evidence, incomplete screenshots for test step level, no steps to reproduce information, lack of understanding on requirements, lack of training” etc. while logging the defects in test management tools. One of the following 4 roots causes needs to be selected for a defect that is identified to be an “Invalid Defects” after the status is updated to “Rejected” according to defect management life cycle:-

1) **Duplicate defects**: - In ST Phase, suppose 1 Release will have 3 cycles to execute in agreement. Now in 1st cycle, tester “A” raised 1 defect related to specific functionality and tester “B” raised another defect related to same functionality what Tester “A” raised earlier. As a result, this defect raised by tester “B” would be treated as “Duplicate defects”

2) **Out of scope**: In few of the cases like banking, telecom applications etc., Testers are very well aware of functionality of the system. Sometimes, they raise the defect related to functionality which is out of the box thinking and not mentioned in the “Business Requirement document”. Developers could reject this defect stating the root cause as “out of scope” for them according to “BRD”

3) **Unable to Recreate**: Most of the times, it has been observed that because of inadequate evidence like “improper screenshots of functionality” related to defect occurred & “lack of information” provided in test management tool(Quality center)by tester’s are the root cause of rejecting the defects, and Dev. team could update the status as ”Unable to Recreate”

4) **Working as Designed** (WAD):- This is another major root cause for rejecting the defects. This happens because of “lack of knowledge on Domain functionality”, “lack of skills”, inadequate and ambiguous requirements, trainings etc. Majorly Developer’s put the defect status as ”Rejected “ and RCA as “WAD”

How to calculate Defect Rejection in percentage?

1) If defect rejected in ST phase, then formula would be

\[ \% \text{ Defect Rejection in ST} = \frac{\text{No of defect rejected in ST}}{\text{Total no of defect detected in ST}} * 100 \]
2) If defect rejected in SIT phase, then formula would be
\[
\text{% Defect Rejection in SIT} = \left( \frac{\text{No of defect rejected in SIT}}{\text{Total no of defect detected in SIT}} \right) \times 100
\]

3) Overall % defect rejected = \left( \frac{\text{Total No of rejected Defect (ST + SIT)}}{\text{Total no of defect detected in (ST + SIT)}} \right) \times 100

**Why do slippages or defect leakage occurs in spite of testing?**

It is very difficult to state that software product is 100% defect free and it will not have any slippages and leakage. There are three main causes that could donate to this reality are:-

**Multiple possible pathways:** There could be many potential paths or combination to be covered in most of the test cycles in different areas. Since today, testing is driven by time and money, it has become very important to plan and spot the high and low risk areas stand on the most likely scenario that could exist in client/user environment and need to originate the test strategy according to avoid high & low risks of slippages.

**Multiple possible user environments:** It is very difficult to follow all possible client environments in one place because of the end users grouping of third party applications, operating system and hardware could be of large range. For example, it is very complicated to test the software which network is merging with many different networks. In this type of networks, it is very difficult to fix the defect at one test phase considering all possible network and software merging together and make sure that leakage would not happen in later stage.

**Multiple possible Inputs:** The multiple possible inputs are generally categorized which includes all the valid and invalid input and the ways the input is key source to the software. The possible source of inputs to the software can be different types like “keyboard, mouse or could be any kind of software or any devices”

**What are the main reasons behind this?**

Below are the main reasons for the slippages and leakage occurring during the testing phase:-

1) **Lack of training:** Lack of training could provide critical during the initial stages of understating design, estimation, building the test strategy and following whole testing process. So training needs to be conducted at the beginning stages if the project even before the test strategy is formulated and this practice needs to be conducted continuously on the project.

2) **Lack of product knowledge & tools:** Product knowledge is also an important role for testing and this practice needs to be continue throughout the project and all the project members needs to be educated about the whole product even though if the project is been executed in modules because the interoperability between the modules will be the critical part of the testing. Tools play the key aspects in testing environment for test suite optimization to execute automation of test cases. Optimization helps to reduce the test and utilization effort and to focus on high risk areas in the testing phases. It also saves the money and time in the project because automated test would cost very less compared to the manual test cases which would provide the complete coverage.
3) **Not adequate coverage:** - The main reason for this would be the lack of coverage which will guide the situation where some areas might go to the customer without tested and the client will end up with very expensive defects. The other area which will be the vigor and revival need certainly more coverage, attention and should be taken care by the experienced hands and imagination of out of box thinking. The best possible solution for handling this is to have effective quality review, 100 % traceability of the design document, best practices, lesson learnt and case studies from the previous releases.

4) **Not capturing the Requirement specification properly:** - For any business areas, it is very important to have the understanding of the requirements given by the customer’s and provide the best possible solutions. If the requirements are not understood properly then it could give the results on providing the wrong estimation and it could impact on planning for lab infrastructure and resource planning, budget & effort, time pressures etc. understanding of the requirement doesn’t mean to get the thoughtful content of the specification but also the dependencies on each applications or functionalities which affect the product and plan accordingly. Another important aspect is “Risk Assessment”, if the requirement is not understood correctly; it will be one of the highly donating factors for failure of the testing and product.

5) **Design document inadequate:** - Design document is the only source for testing team to prepare the test strategy which includes the clear picture on limitations, facts documented correctly. Any ambiguity on design document or any assumptions taken by testing team which would be the source of possible slippages. So the testing team needs to be proactive and participate in design document reviews to make sure that document is correct and clear.

6) **Lack of effective peer review of test cases & quality reviews:** - Review of the test plan plays an important role for developing the high quality of test plan which would certainly result in finding the high ratio of number of bugs in terms with number of test cases. So it will include the domain experts in review panel which would provide the ideas which could be missed out and prove critical slippages. Also testing experts in the review panel will help and make sure that test cases are correct, clear and informative covering the business requirements and rules, importantly with the pass/fail criteria for the tester’s to decide the status and outcome of the test cases. Any ambiguity or even a slightest confusion in the test cases would result the assumption of tester’s and a prospective defect could be missed out or treated as a slippages.

7) **Ignoring sporadic occurrence problems:** - This reason is the another important factor that experienced tester’s would validate an issue as one time occurrence or sporadic and kept an issue under observation. The observation will not occur again and might slip during testing phase without giving any attention. As a result this will come up and reproducing in the client site which would be very costly to fix because the problem is sporadic and not dupable. So it will be very important to report one time occurrence or sporadic problems during testing phase and it will be very less cost to fix at this stage.

8) **Change in the code due to defect fix:** - This activity could happen in the middle of the testing phase due to implementation of new content or changes due to bug fixes. Change in the code would provide increasing of testing effort, so it is very important to keep the focal point and trail the normal process right from developing test strategy to test plan so that the coverage is optimized and then reviewed in spite of tight schedule. Reviews should not be skipping at this point of time.

9) **Tester’s fault:** - Most of the tester’s fault would be the lack of product knowledge, not enough clarity on test cases written and not following the basic rules for testing principles. If the testers would have proper documentation and product knowledge then that would be clear on the expected behavior for each action addressed and that would reduce the slippages.

10) **Lack of coordination between design and test team:** - it is very important for the test team to work closely with design team to get the understanding on the limitation of the design document and how it would impact the end user directly or indirectly and raise the change request at any point of time. it should be done having the brainstorming sessions with test leads, test managers, design and dev. team and the review meeting which would provide some trace to identify the problem areas for the frequent changes done in design document.

**Why severity and priority are the most important aspects for defects and its hidden perceptive?**

The hidden perception about severity and priority of the defect are always confounded while logging the defect in test management tool (QC) that “what would be the appropriate severity and priority of the valid problem needs to be assigned”. Severity and priority of the problem are the most important aspect of the defect management life cycle which will help to analyse the metrics and the importance of business. Here we will explain the classification of severity and priority with example which would help to better understand the difference between two.
Severity: - it relates to technical impact and it indicates how bad the bug is and reflects its impact to the product and to the user. Severity is the asset for testers.
S1:- Data loss, Data corruption and System crash
S2:- Loss of Functionality, Incorrect Result
S3:- Rare Occurrence, UI Layout

Priority: - It relates to business impact and it indicates how important is to fix the bug and when it should be fixing. Priority is the asset for business stakeholders.
P1:- Immediate Fix, Blocks further testing
P2:- Must fix before the product is going to release
P3:- Should fix if the time permits

Examples of High Severity (S1) and Low priority (P3) bugs: - Suppose Company “A” is the manufacturing company of the VOIP phones and delivered 100 phones to company “B”.100 phones are used by 100 individual user’s in company “B”.99 phones are working fine but one user is impacted because he is not able to dial the 10 digit mobile number. While dialling the 10 digit number, it is accepting only 9 digits. All the digits are written well as in numbering. While reporting to company “A”, it would be treated as high severity (S1) and low priority (P3) problem. Status of defect “Deferred” would also be a good example of high severity (S1) and low priority (P3).

Examples of High Severity (S1) and High priority (P1) bugs: - In Google search engine, if user enters some valid text in typing box and clicked on “search” button and application crashes then it would be treated as high severity (S1) and high priority (P1) problem found; because most of the users are impacted and the business for Google is also impacted. Another example could be if the user selects the report to run as in “PDF” format, and run the report accordingly. As a result it provides the “Reporting Service Error” with no content displayed; it would be treated as high severity (S1) and high priority (P1) problem.

Examples of High priority (P1) and Low Severity (S3) bugs: - Suppose a customer would like to buy a new car from the showroom, checked the tyres, started the engine and check the specification but at one point observed that there are some scratches on the front door outside of the gate, the question here is will he buy? Not obviously many of the customers will see that scratches and will not buy. Here the business is impacted, so the priority would be high (P1) and severity would be (S3) because there is no technical impact.

Examples of Low priority (P3) and Low Severity (S3) bugs: - These types of error would be cosmetic errors like error message displayed “To date should be gReater than end daTe”. Here the cosmetic errors severity would be (S3) and priority would be (P3) while testing the error messages need to be displayed because already error message is displayed but there is some font issues occurred.

3. How to implement effective test process improvement and test execution & tracking to minimize the cost of slippages, defect rejection

Static Testing of Business Requirements and Rules: - This needs to be start from the JAD (Joint Application Development) sessions where the entire stakeholder’s come together and agree and negotiate upon requirements. In this session, all the stakeholder’s discuss on requirements (complexity, business flow etc.) and attending this kind of sessions are very useful for the testers at initial stage for the requirement analysis. Testers could understand the business flow; how it works and how the functions related to the requirements would implement. This kind of sessions would provide the great helps to the testers to perform static analysis for the requirements and business rules. There are several tools available for the business requirements and rules like, IBM Req Pro, CBRM (Central Business Rule Management) etc.

Static Testing is a detailed assessment of a work product’s characteristics to an expected set of attributes, experiences, and standards. Discovering discrepancy or defects, early in a project is aimed
Implementation of Effective Test Process Improvement and Test Execution to minimize the Cost of “Slippages”, “Defect Rejection” related to Client Deliverables

at producing a “defect free” work product, before it is passed on to the next phase of the development process. Inspections which are the most disciplined form of static testing continue to be one of the most economical and productive methods for detecting defect early within the product development life cycle. Some other representative examples of static testing are walkthrough, desk checks, prototyping, reviews, checklists, mapping, usability techniques and audits.

**How to perform Static testing?**

Testers will perform reviews to application parent and children Business requirements (BUSRs) and Rules (BRULs) to ensure all the specific needs, operations, definitions, and constraints associated to the application under test are well defined and unambiguous so they can translate into scenarios or cases and be traceable for coverage.

**Examples of ambiguous requirement:-**

**Req 1:-** Application will support various clients in single sessions.

**Static Result:** - Above requirement must specific the number and type of clients where the application will support

**Req 2:-** Change one time password and verification code for contact detail

**Req 2 BRUL A:** -One time verification code and password needs to be send by SMS on contact number mentioned when clicked on submit button

**Static Result:** - Above requirement and business rules must specify the type of contact number. Because if the contact number is landline phone then here business rule will not work

**Static Test Plan:** - Static test plan describes the use of static techniques to verify the correctness and completeness of all the development and test work products against precursor work products, business, organization standards and technical expertise.

**Static testing is achieved by:-**

- Finding and removing any defects in the work products, as close to the source and early as possible in the development life cycle
- Reducing the development and testing life cycle time by minimizing the rework

It will contribute extensively to improve the quality of the development/testing work products.

**Objective:** - Requirements comprise statements of simply what the business/customer functional and non-functional wants and needs. This is usually expressed in high level of what the business requires, rather than specific process or functions the system may perform. Specific process or function descriptions are defined in the system requirements. We should verify the requirement is testable; free from ambiguity and no spelling/grammatical error.

**A simple check of “problem” words and phrases within the Requirements Specification can eradicate a great amount of confusion.**

- Incomplete IF… THEN statements- we should look for the statements like these that don’t have an ELSE condition. we should ask our self that what will happen if the “IF” doesn’t happen
- User Friendly, Cheap, Fast, Stable, Good, Efficient, and Small- These words are not testable because they can’t be quantified. It must be further defined
- Etc., And so forth, And so on, Such as- Lists that end with these terms can’t be fully tested. it creates confusion because they are open ended and open to assumption
- Most, Mostly, Some, Often, Usually, Ordinarily, Customarily, Sometimes- These words are too hazy to be properly tested
- Customarily, Most, Mostly, Therefore, Clearly, Obviously, Certainly-These words attempt to categorize or convince the tester to accept something as given. This is a kind of trap ,don’t fall into it

**Key points to remember for accurate approach to static testing for system level requirements:** -

- Each requirement addresses one and only one item, needs to be unique and stands alone from all others
Each requirement is complete, fully describing the event, condition, action or activity and has a unique id. It is described using business terminology and is free from subjective or imprecise terms such as fast, user friendly, easy to use

Each requirement is accurate; dealing with amounts, quantities, time, distance, volume, currencies or other dimension must clearly define the unit of measure used and must use a unit of measure consistent with all other requirements dealing with the same dimension

It defines the calculation logic or formula to use to achieve desired results; and it describes a quantifiable condition, activity, or action that leads to an equally quantifiable result

Each requirement can be implemented within the time schedule, budget, and available technology. It will contribute more to realizing the expected business value than the cost of implementing it and is legal

Each requirement is described in concise terms that leave no room for interpretation and it describes the condition, events or actions clearly and concisely so that the same inputs will consistently produce the same results

It will identify all condition both positive and negative that the requirement is expected to address

Each requirement has an acceptance criteria that aligns with expected business value and converted to a specific test condition provide sufficient information to define an accurate, complete and unique test condition

Each requirement is fully traceable from key business features to test cases for each test phase and the impact of a change to any requirement, design specs or code to requirements, design specs can be identified and assessed

**Requirement traceability verification matrix (RTVM):** This is to trace the requirement and its business rules against the test cases and its iterations for execution.

<table>
<thead>
<tr>
<th>GUI Location</th>
<th>GUI Functionality</th>
<th>Sub level</th>
<th>Status</th>
<th>Inquiry / Maintenance</th>
<th>Variation</th>
<th>Scenario</th>
<th>Business Requirement</th>
<th>BRUL (Business Rules) / Assumption</th>
<th>Test Coverage</th>
<th>Iterations</th>
<th>Test Data</th>
<th>Pop Up messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Screen</td>
<td>Personal Screen</td>
<td>SRS / FCR</td>
<td>Inquiry</td>
<td>Data Set</td>
<td>mobile</td>
<td>update the sms functionality for GUI functionality</td>
<td>BUSR 001</td>
<td>BRUL ABC</td>
<td>1</td>
<td>5</td>
<td>CD accounts</td>
<td>BRUL 123</td>
</tr>
</tbody>
</table>

**How to write effective test cases using parameterization and modularity concepts in Quality Center (Test Plan)?**

It can be expanded the scope of process for business and its component defining the component parameter that a component can either receive or return. This process is known as “parameterization”. It increases the power and elasticity of the tests and its components. A component parameter is an element within the business component that can be assigned various values. Input and Output of the component parameter allows component to use variables values that affect the test result and pass values between components in the business process tests. Generally we use the parameterization concept to write the complex test cases where we have to pass the various fields value in a single screen. Modularity concept is defined as when we are writing the generic test cases and it needs to be executed in some of the test cases. We can treat this as “Call to Test” in “Design Steps”. Mostly, generic test cases regarding functionality needs to be placed in “Global” module in test plan and when we are writing the test cases; could use “Call to Test” from design steps for particular “Description” of the steps.

**Naming convention of the test cases in test plan and its details:**

BUSR001_BRULABC_GUI_To Verify the field in personal detail screen

BUSR001_BRULABC_DV_To Validate the field value for personal detail tab

BUSR001_BRULABC_FUNC_To Validate the functionality of field value in personal detail tab

BUSR001_BRULABC_BATCH_To Validate the field value is updated after daily job run

Here 1st statement says about to check the GUI screen in personal details tab where we verify all the fields are present according to business requirements and business rules. Here, BUSR001 is treated as “business requirement” and BRULABC is treated as “business rule”; 001 is treated as business requirement id and ABC is defined as business rule id associated to requirement 001. 2nd statement defines the DV (Data Validation) where we use to check whether field value in personal detail tab is filled with data or not. 3rd statement talks about to check the functionality of each field value or tag in personal detail tab like SMS.
functionality check in mobile number or address, email id etc. details. 4th statement defines the daily batch run if the personal details are updated after that it should display successfully. Here, we see some of the screenshot how we do the naming convention in QC:-

Fig1: Test Plan Detail tab

In Test Plan Detail Tab, see the description section; we can provide the detailed information of the test cases like:-

**Objective:** - To validate the functionality of field value in personal detail tab

**Test Data:**
- 1) Checking account:-1234567897
- 2) Savings account:-6767676767
- 3) Money Market account:- 1525678509

**Iteration or instances:** - 3

**Requirement description:**
- BUSR001:- Update the personal detail information in client profile screen for all types of account
- BRULabc:- Check SMS functionality on mobile/landline number and pop up message displayed
- BRUL123:- Pop up message is displayed as “Message Sent Successfully”

**Parameters:** - Username, Password, Address, Mobile Number, Phone Number

**Comments:**

*Note:* Iteration and instances talks about how many times we will execute this particular test cases, so accordingly in test lab we have to pull 3 times to execute this test cases. Parameters defines what value we need to pass through test cases

**Status of the Test cases in Detail tab:** - Here, we are using the status of the test cases:-

1) **Design:** - If we are writing the test cases then status should be always in “Design” mode
2) **Imported:** - If we are importing the test cases through excel sheet into test plan tab then status needs to updated as “Imported” mode
3) **Inactive:** - It describes if the test cases in “Inactive” mode where we are not going to execute in test lab
4) **Obsolete:** - It states if the test cases are outdated and new business requirement and rules are updated, based on that needs to write new test cases
5) **Repair:** - If the concern testers are modifying the test cases once it was in “Ready” mode then the status should be set as “Repair”
6) **Ready**- Ideally this status needs to be set by “Business Analyst”, once they confirms that all the test cases are covering Business Requirements and Business Rules and after providing the “understanding on the functionality” or “BA Deck prepared” by tester’s and based on satisfaction “BA” can update the test case status as “Ready”. Once the status is set as “Ready” by “BA” then testers can pull the test cases in test lab.

In Test Plan Design Steps Tab, Testers can add “Parameters” by clicking on <P+> button while designing the steps, once clicked on <P+> button, it will ask to add “Parameter Name “under “Parameter Properties” or directly typing the value to insert as following naming convention as <<<value>>>; For e.g. like <<<Mobile Number>>> or <<<username>>> or <<<phone number>>> or <<<address>>> while covering the test scenario in particular test steps. By clicking on option <P>, it will show the current test parameter’s what tester’s added successfully like username, password, address, mobile number, phone number.

![Fig2: Test Plan Design Step tab](image)

![Fig3: “Call to Test” in Design Step tab](image)
Challenges’ faced while writing the test case prep:-

External dependencies
- Business Requirements are ambiguous or not specific so it is very difficult to write the scenario’s for each independent business requirements and rules
- Query/Issue log is sent to BA/SME’s for clarification on business requirements or business rules conflict, so test case coverage and complete traceability are inadequate

Process dependencies
- Could increase the number of CR’s if static testing of the business requirements and rules are not done at the time of requirement gathering
- Work pressure on resources while writing the test cases related to CR’s at the time of test case execution phase
- More rounds of peer review can become less if the timelines are too short
- Peer Review comments within internal team members based on the input given or rework and action items needs to be corrected before doing the preparation and presentation of BA deck

Key points to remember while preparing the test cases in Test Plan:-
- It is very important for the tester’s to get the clarification’s on ambiguous business requirements and business rules conflict from BA/SME’s/Client
- All the resolved query or issue log done by BA/SME’s/Client needs to be place in common repository, so others can see and refer in future for the clarification done and could map with upcoming requirements associated with this
- Static testing of the requirements needs to be conducted once the tester’s receive the FCR/BRD/SRS, so it will minimize the counts of CR’s at the time of test case execution phase or in later releases
- Be always sure that peer reviews comments based on logical design, linkage requirements, expected results, business rules missing are corrected and closed on high priority
- All test data needs to be identified or requested related to complex or normal test cases based on high priority before the test case execution starts
- Make sure that wireframes of the screen are present in FCR’s or BRD related to functionality; if not then request for availability
- It becomes very necessary to provide the description and expected results with detailed information while writing the test cases and in generic format so at the time of execution peers can read, understand & able to execute
- Be specific related to the current test parameters what testers are going to add in test steps while designing the test cases. It should pass the parameter’s value and executable from test lab
- Step design should not be more, try to use “call functions to test “ to pull generic steps from global module where generic test cases are placed
- Tester’s needs to cover all the scenario prepared in RTVM sheet and should be aligned & mapped with test cases

Advantages and Disadvantages:-
- Test case preparation count would be less and it will cover the fixed field to pass the parameters values in it
- Tester’s will get the more rounds of peer review within test case preparation timelines and action will be address according to it
- Quality of test cases will improve and knowledge of other team members would be more
- Once BA decks are prepared and presented in front of stakeholder’s ; it would provide the test coverage related to the business requirements for particular functionality
- Timeframe limitation of peer review is short if tight scheduled deadlines for testing and availability of resources to conduct peer reviews done for Test case preparation

Peer Review Template Conducted for Test Case Preparation:-
How to perform test case execution with readiness for functional requirements with effective tracking?: Once the test case preparation is done by testers and status is set as “Ready in “Test Plan” tab; then it is allowed to pull how many iterations or instances needs to be executed in test lab by clicking on “Select Tests” from “Test Lab” tab. When selecting the appropriate test cases from “Test Plan Tree”, it will ask you to provide the “Parameters” value to pass for 1st iteration of test case for execution or how much iteration of test cases to execute. Testers can also see the no of iterations or instances have been pulled for execution like [1], [2], [3] etc. in “Test Lab” tab. By double clicking on particular test cases, Testers can identify the parameter’s value from “Configuration” tab which was given at the time of pulling test cases in test lab and from “Details” tab (Fig 4) can identify what is the objective of test cases, information related to test data and parameters name at the time of writing the test cases in “Test Plan” tab.
We can add two new columns in “Execution Grid” from “Test Lab” tab as “Execution Readiness” to conduct effective execution & tracking for the test cases and “Comments”. “Execution Readiness” has different dropdown values as: -

1) Blocked-Defects
2) Blocked-No Code
3) Blocked-Other
4) Executing
5) Hold-Data
6) Hold-Batch
7) Hold-Environment
8) Ready to test

1) **Blocked-Defects**: - When some defects are raised for particular/specific test case and other test cases are impacted or have a dependency on this particular/specific test case. Here, Testers can put this value as “Blocked-Defects” for the impacted test cases and link the defects with it. Once the defects are “Fixed” and “Ready for Retest” then testers will have to execute particular/specific test case plus impacted test cases. Testers can update the “comment” section based on status set

2) **Blocked-No code**: - This status could be set once the code is not delivered in particular release for FCR’s or functionality and the other test cases are impacted due to code or have a dependency on this would be set as “Blocked-No code”. Testers can update the “comment” section based on status set

3) **Blocked-Other**: - Basically this status would be use when the query/issue’s has been sent for clarification or resolution or for any other dependency. Before the confirmation comes testers could set this status as “Blocked-Other” for particular test cases and its dependent test cases. Testers can update the “comment” section based on status set

4) **Executing**: - This is for when testers are executing test cases; at the time of pulling real time execution progress report and any testers are executing test cases then report would show this particular test cases are in progress for execution. So based on this report Test Leads/Managers could get to know the real time execution status for the test cases. Testers can update the “comment” section based on status set

5) **Hold-Data**: - If any test data is not available at the time of execution or data is corrupted so testers can set this status as “Hold-Data”. So called this particular test case is having dependency on particular test data for particular FCR’s or functionality. Testers can update the “comment” section based on status set

6) **Hold-Batch**: - This is related to scheduler run or batch run for particular functionality having dependency on the test cases. Probably input has been given and waiting for the batch run like daily, weekly, fortnightly, or monthly, then in that case testers could select this status as “Hold-Batch” because of batch run dependency, so based on this situation testers can plan accordingly. Testers can update the “comment” section based on status set

7) **Hold-Environment**: - This is particularly dedicated to unavailability of “test environment”. It can be treated as environment is down or environment dependency accordingly needs to update the comment section. Testers can update the “comment” section based on status set

8) **Ready to test**: - Once the test cases are ready to execute then assign the particular testers for execution in “Execution Grid” and set the status as “Ready to test”. It means that this test case is having availability of test data, environment etc. Testers can update the “comment” section based on status set

---

**Fig 5**: Test Lab (Test Instance Properties)
Implementation of Effective Test Process Improvement and Test Execution to minimize the Cost of “Slippages”, "Defect Rejection” related to Client Deliverables

Challenges’ faced while executing the test case:-

External dependencies
- CR’s (Change Request) is implemented in particular release at the time of test case execution phase to deliver within current time frame
- Dependency on the queries sent for the clarification to BA’s / SME’s on FCR’s/Functionality
- Test case preparation for CR’s at the time of execution phase if it is deliverable in current release; faced the challenges while doing communication with onsite folks for concern functionality owner
- Unavailability of test environment and if any dependent environment is down so it would impact on execution timeline

Process dependencies
- If one test case is having 20 steps to execute; it has been observed that testers raised the defect (which is no blocker for next step) in between like on step 12 and rest of remaining steps are showing as “No Run”, it might have some possibility of getting more defects from step 13 to 20 in current iteration/instance execution
- Defects raised at test step level doesn’t have proper attachments and Business Requirements/Rule id or detailed information not mentioned, so it might increase the defect rejection ratio
- Dependency on test data if it is not available or not found before test case execution starts, it might impact on execution timelines for high priority of test cases
- Resource dependencies and its KT plan

Key points to remember while executing the test cases in Test Lab:-
- If CR’s have been implemented related to functionalities at the time of test case execution phase to deliver within current time frame then testers needs to contact immediately with function owner and ask for the detail information of particular CR’s impacting the functionality for an applications. In this case, testers would have gain of knowing the CR’s functionality and would be able to prepare the effective test cases with the help of function owner
- Follow up with BA’s and SME’s on query sent at the time of execution and needs to update the issue facing on daily status call with client, so they could discuss with BA’s/ SME’s and provide the resolutions immediately
- Need to have daily status call with onsite folks, clients and defect managers in war room so they could discuss on availability of environment on time and if environment is down for sometimes then 24/7 support from operations team to fix this so testing timelines would not impact because of this
- Testers need to execute all the test steps if the defect is found in between and has no blocker for next steps. It will save the impact on build delays which would be providing to testing team
- It is very important for the testers to provide all possible information like summary, detailed description, severity and priority, steps to recreate, attachments, comments, parameters value, business requirement/rule id etc while raising the new defects at test step levels. Providing as detailed information would save and minimize the defect rejection ratio.
- It’s better for the testers before raising the defects, they need to talk to concern developer’s or raise the clarification to BA/SME/Dev regarding functionality issue or understanding on issue found, so it would decrease the impact and its minimization of “Defect Rejection Ratio”
- For high priority of test cases, test data is very important factor, it would be great if the testers would raise the request for test data or find at the time of test case preparation phase or at least 10 days before the actual test case execution starts, so there would be less impact on execution timelines and its test data dependency
- Testers also needs to find the test data related to test cases parallel which are on high priority for execution in current time frame
**Defects:** Identifying and logging a new defect is an important activity in complete STLC. While raising a new defect in QC, considerable details should be provided to minimize the defect rejection ratio. QC fields should be customized to provide detailed and accurate information to programmers. Following are the major field which mandatorily should be populated not only to reduce the defect rejection but reduce the further cycles of communication among developers and testers which definitely saves time and reduces frustration in tight schedules of projects:

1. Summary
2. Description
3. Detected on Date
4. Severity
5. Priority
6. Release
7. Cycle
8. Detected in Build
9. Detected by
10. Assigned to
11. Defect slink from

**Summary:** Writing a one line defect summary is an art. Every tester should master it. Defect summary is the first thing that Programmer, Project Manager or other stakeholders read carefully to understand the defect. A well-written summary helps the stakeholders to quickly understand the defect. Good summary should reveal the severity of defect and should communicate the problem statement only. Most of the times programmers, if they are not able to understand the defect from summary, they straightway reject the defect. Summary is a separate field than description, which usually is misunderstood by testing community.

**Description:** Description provides the details about the defect. It elaborates the summary of defect. While phrasing description, testers should keep in mind that they have to spoon-feed the details to developers. Good written communication skills definitely add value here. Following the description paragraph below details should be added. Headings in bold should be maintained:

**Steps to Reproduce:** Tester should provide step by step details that how that particular defect can be reproduced by the developers. These steps should be numbered items and sequence of steps should be same which tester has followed to get that defect.

**Alternative Steps:** If tester is able to reproduce the defect by following alternative steps, which should be communicated under this head. This can add value and help developers to understand the defect in detail.

**Expected Result:** Provide the details of result which was expected after following the steps to reproduce the defect.

**Actual Result:** Provide the details about actual behavior of application after following the steps to reproduce the defect.

**Additional comments:** Any other details/observations can be provided here. Reference to Requirements Document/Design Document/SRS can be provided here which

**Screenshots/Attachments:** This is another most important item and should not be missed. Here testers should provide the complete screen shots starting from first steps, rather than the only of screen where defect got identified.

**Detected on Date:** This is another important field, which helps to track defect ageing and can be traced back to version of build in which the defect was identified.

**Severity:** No doubt along with Priority, this is the most discussed field in testing community. Severity is nothing but the seriousness of the problem. How it is impacting the functionality of application under test or other integrated application/s. There is no hard and fast rule to determine the severity of defect but rating scale is must, which in QC is 1 (Most severe) to 3 (Cosmetic).

**Priority:** This should be set by the Test Manager/Project Manager.
Defect slink from: This is not only an important field but a metrics in itself which provides the number of defects which slinked from one phase to another phase of testing. Defect slink from field drop down items should be:

1) ST
2) SIT
3) UAT

Challenges faced:

- It is hard to analyze and select the appropriate value for ‘Defect slink from’ field
- It is a pain to correlate the defect type value with “defect slink from” field value “Requirement, Data, Interface

Key points to remember:

- During functional testing, defect found in ST should not be consider as slinked from Unit Testing
- If it is a functional defect in SIT or UAT and it is a slink from ST then the defect type would be selected as “Functional”
- If it is a workflow/data flow defect in UAT and it is a slink from SIT then the defect type would be treated as “interface”
- If it is an data corruption issue defect in SIT or UAT and it is a slink from ST then the defect type would be treated as “interface”

4. Conclusion

Following point’s needs to be taken care by the test manager, test lead and the tester to minimise the risks of slippages on the software testing is quite possible

- Testers need to analyze and identifying the root cause for previous release done for that area which need more coverage and ensure that this is taken care for as a part review activity
- Make sure that coordination has been set up with design and test team to understand and identifying the low and high risk areas and then plan the test strategy accordingly
- Always have to pay concentration on all the problems even it is one time occurrence or intermittent
- Make sure the team has the required product knowledge and also plan for regular training session to keep the team updated about the day today developments of the overall project
- Even it is quoted as design intent as and when required, testers need to raise the concern regarding this
- Try to imitate the customer environment when planning for the lab and equipments.
- Testers need to use the tools where able it is applicable to automate to reduce the effort and the saved effort can be utilized in different areas
- Ensure that testers need to refer the approved, completed and baselined design documents to understand the expected outcome of each action to make sure that there is no assumption taken in the expected result section
- Develop the test plan based on the test strategy with the transparency on the test cases
- Strengthen the review process to focus on 100% traceability to the design document and optimized coverage for the testing.
- Make sure that testers need to follow the detailed process even it is a very small piece of code churn, they need not to follow the shortcuts
5. References

- HP Quality Center
- Software Testing (Ron Patton)

6. Author’s Biography

6.1 Sharad Kumar is an engineering graduate in electronics field from North Maharashtra University & completed MS in Software Systems from BITS, Pilani (Rajasthan). He has total experience of 7.5+ yrs in software testing & currently working as a Senior Consultant with Capgemini India Pvt Ltd, Bangalore. Previously worked with IBM, Deloitte consulting; Aloha Technology. In previous engagement he was involved in preparing best practices, case studies, ideas & innovation and in process areas like effectively participated in the organization as a part of CMMI v 1.2 to 1.3 assessment preparations; was member of Testing COE and internally checking health of the project

6.2 Amrik Singh is an MCA from MDU Rohtak & completed QTP, QC, Load Runner, ISTQB certification. He has total experience of around 10 yrs. in software testing & currently working as a Senior Consultant with Capgemini India Pvt Ltd, Bangalore. Earlier he worked with ANZ, UHG, and Logica. In previous engagement he was involved in extensive experience in Automation; performance testing was actively involved in setup of TCOE. he has hands on experience in agile, waterfall Model