INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

Test Strategy And Integration Test Plan For Smart Meter Implementation Program

Anubha Chauhan
Research Scholar ,Suresh Gyan Vihar University,vJaipur,Rajasthan, India
Dr. Naveen Hemrajani
Professor,Suresh Gyan Vihar University,vJaipur,Rajasthan, India

Abstract:

A testing methodology is a tool or method used to test an application. A testing strategy, on the other hand, is a holistic view to how you will test a product; it's the approach you will take, the tools (and methodologies) you will use to deliver the highest possible quality at the end of a project. In software quality, the test strategy consists of a myriad of methodologies, activities, and staffing solutions. The strategy overall sets the acceptable bar and calls out how the test team will achieve that bar. It is the sum of all the inputs, in an organized plan. Testing methodologies are the different approaches you will take to testing. The purpose of this paper is to define the strategy, procedures, and tools for testing the technology enablement of the smart meter implementation program. All phases of the testing lifecycle will be covered, from Unit Testing to User Acceptance Testing. However, the primary focus is on Integration Testing of System-to-System Interfaces, End-to-End business transactions and User Acceptance Testing. As such, the scope of integration testing includes transactions that start or terminate at vendor systems or field-devices. Then described is approach that provides assurance the release will enable the required integration interactions between the applications in order to facilitate and support "real-life" business transactions. This lifecycle starts with the application or system specific tests performed by application developers and infrastructure engineers (unit / string tests) and does not end until the business owners have given their approvals after user acceptance testing. The focus of integration testing is to verify that the applications that are linked together successfully enable "real-life" business transactions.

Keywords*: Testing, Testing strategy, Test Lifecycle, Software testing, Integration Testing*

1.Introduction

The objective of the test strategy is ensure the business owners have a well-deserved confidence that the systems can work together to support real-life business transactions. This strategy is designed to account for the complexity associated with having to integrate multiple new applications into client's core customer and billing systems. These applications are being developed by different teams, on different platforms, using different tools and development approaches. Our strategy covers the complete test lifecycle. This lifecycle starts with the first, informal tests done by application programmers (unit tests) and ends when the business owners have given their approvals after user acceptance testing. However, the strategy's focus is on integration testing, where all the applications are linked together to verify they can support real-life business transactions.

This strategy is designed to account for the complexity associated with having to integrate multiple new applications into client's core customer and billing systems. These applications are being developed by different teams, on different platforms, using different tools and development approaches.

Our strategy covers the complete test lifecycle. This lifecycle starts with the first, informal tests done by application programmers (unit tests) and ends when the business owners have given their approvals after user acceptance testing [1]. However, the strategy's focus is on integration testing, where all the applications are linked together to verify they can support real-life business transactions. As such, the scope of integration testing includes transactions that start or terminate at vendor systems or field-devices.

1.1.Test Lifecycle

Figure1.1 shows a view of the lifecycle and team responsibilities. This diagram outlines two key aspects of the test strategy. First, it outlines responsibilities of the various teams and organizations involved in testing. Second, it outlines a comprehensive set of activities across time. These test activities start with test planning, go through Application and Integration testing, and end with Scalability, UAT, and Security testing.

At a high-level, here's the lifecycle of testing covered by this document:

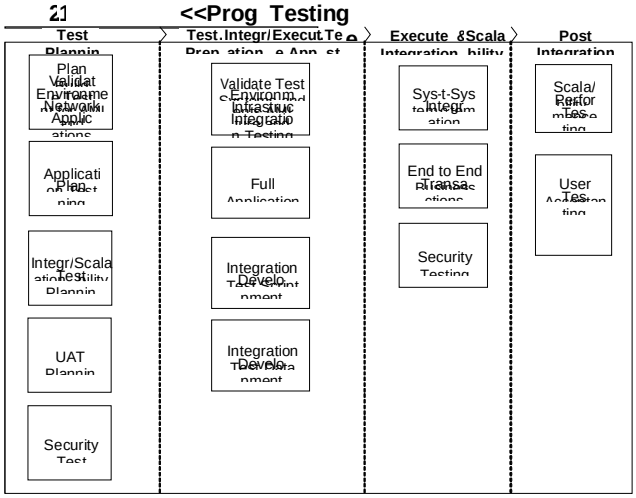

Figure 1.1: Test Lifecycle

Step 1: Plan the effort

- The individual teams plan how to test their applications and new development prior to handing off for integration testing.
- Installation and technology vendors set up and validate their test systems to support transactions that will be sent or received by the systems.
- Client Smart Meter Operations team creates and validates a representative environment of electric meters and network components for subsequent integration testing with back office applications.

Step 2: Prep for integration testing

- Application teams develop their applications. Then they thoroughly test them to ensure their new code within their systems is solid. Application teams in this case refer to the groups responsible for the Legacy Applications .The Integration Testing Team develops and finalizes the test cases to verify data exchanges spanning multiple systems are supported.
- Interfaces between the applications (once the source & target systems and the integration between them are individually tested and ready) are string tested. This is a combined effort between the application teams. This is similar to unit testing of the interface, but is designed to validate two applications exchange data across the integration layer. Mock data may be used to test transfer of transactions from source to target systems.

Step 3: Integration and Scalability Testing

As applications and integrations are developed, they are brought together for a full range of tests

- Here the Integration Testing Team validates that applications can send data back and forth. However, unlike string testing, integration testing is formal, rigorous, and extensive. Integration testing starts by testing system-to-system data exchanges. Once enough interfaces are delivered to support an end-to-end business transaction, the integration test team verifies that a full range of business transactions can flow correctly and without interruption from a source system to its end points. These end-to-end transactions often flow across many systems, and each step of the way is monitored and verified.
- Once the code passes agreed upon acceptance criteria, scalability tests are executed. The ability of systems and interfaces to handle real-life loads together is verified. Test loads are simulated and transactions simulating end-to-end business flows are executed.

Step 4: Post Integration Testing

The final testing includes:

User Acceptance Testing (UAT), where business owners and their representatives run a series of tests to independently verify that the systems will handle real world transaction. A sub-set of the end-to-end business transactions may be used for this.

Both aspects of the test lifecycle – test activities and test team responsibilities are described below.

1.1.1 Test Teams Responsibilities

Meter Technology / AMI Network Teams Creates and administers a test-environment of meters and networking infrastructure upon which String (Interface unit), Integration, UAT, and Security testing can be conducted. This lab environment simulates the types of meters that will be deployed in the field. Application Teams (MDMS, ADCS, Customer Information System, Billing System Teams Application teams) need to rigorously and fully test their applications through their normal lifecycle before releasing the applications for string testing. Developers of an interface or interaction between systems must unit-test their code or configuration [2]. This team also conducts string

testing to verify application-to-application communication before handing the integrations off to the integration test team. Integration Test Team Organizes and executes formal testing of system-to-system interfaces and end-to-end business transactions across applications. They coordinate the execution of the integration testing and end-to-end transaction by working with the business. Performance / Scalability Test Team Part of the integration test team, this team verifies the overall integrated solution can handle real life loads and meet business-defined performance standards under the highest-expected transaction volumes. User Acceptance Testing Team Integration Test Team with support from client's Business Process team will execute a set of tests to demonstrate that the systems can handle the full range of data transactions seen during daily operations. Independent of the integration test team, the UAT team represents and reports to the business owners. The UAT team will be tightly integrated with the "Integration Test Team". Security Test Team The Security Team verifies that systems and the infrastructure are protected from unauthorized access to the applications and sensitive data.

1.1.2 Test Phases And Activities Within Phase

The testing lifecycle identifies four major phases, each with its own test activities

1.1.2.1.Test Planning

Test planning includes two key deliverables: test strategy and test plans. The Integration Test Strategy, which is this document. More detailed than the test strategies, test plans describe the tactical aspects of testing. Each test plan lists specific functions to be tested and describes how these tests will be organized. It also outlines required data for each set of tests.

1.1.2.2.Preparation For Integration Testing

During this phase, applications and integrations are developed and tested individually. The following describes the basic activities performed during this phase. Unit testing is at the lowest level and the least formal of testing phases. Unit testing will be performed by programmers to validate code before handing the code off for rigorous, independent testing [5]. Application testing a full set of tests to verify an application meets its business requirements and is ready to integrate with other applications to support end-to-end business transactions. Script Development of sets of steps and expected results by

an independent tester to validate that integration between systems meets the business requirements. Test scripts are also developed to validate a set of applications and integrations can support an end-to-end business transaction. Test scripts include step-by-step instructions and expected results. Test Data Development This task includes the design, creation and set-up of test data that will be used for the execution of test scripts.

1.1.2.3.Integration Testing And Security Testing

During this phase, we verify applications and integration-layer code can work together to support end-to-end transactions. In addition, during this phase security specialists run exhaustive security tests. Testing efforts that make up this phase are: String Testing to verify that one application can communicate with another application: similar to unit testing but specific to a system-to-system interaction. Integration Testing is the point at which applications and integration code are brought together for testing by independent testers. Integration testing is rigorous, comprehensive, and formal. For the program, integration testing has two components System to System Integration Testing these tests are started as soon as the three pieces of the equation are completed and string tested – the source application, the integration, and the target application. Tests are based on the business requirements provided for the system to system interaction .End-to-end business transactions across applications End-to-end testing is conducted when enough interfaces are delivered and validated to support an end-to-end business transaction. A test of a business transaction is meant to validate the applications can work together to support a real-life business process. Often these data transactions will start or terminate at either a meter or a vendor system. An example of a business transaction is the initiation of a DR event. This business transaction starts when the DR event is called in the Demand Response System and completes when the event information is propagated to the MDMS, DR Customer Notification System and Customer Information System.

1.1.2.4.Post Integration Testing

When Integration Testing Exit Criteria have been met to instill high confidence in an interface or a business transaction, the final test activities will begin. These tests are completed in an environment designed to simulate the final production environment. Scalability Testing This phase of testing focuses on the ability of an interface to handle a high volume of data transactions. Test scripts for this phase are designed to simulate the heaviest transaction volume that could be expected across an interface. The objective is

to verify that the interface can handle the highest expected data loads. User Acceptance Testing_UAT is the final functional test. It is a set of tests selected and executed by client to validate that the applications and the integration can all work together to support real-world business transactions. Often UAT consists of a subset of end-to-end integration tests. UAT data and scenarios are selected by representatives of the business owners to give business owners high-confidence that the systems will be able to support their core business transactions.

2.Test Strategy Fundamentals

Our test strategy is based on these working principles and assumptions. The application teams may use whatever test tools and procedures they prefer, as long as the test cases and execution results are documented and are available for review in a centralized location. Network traffic validation and sizing will be performed as part of network testing. The application and technology vendor teams will plan for and test their own applications. In addition, they will participate in System-to-System, End-to-End Integration, User Acceptance and Production Readiness testing with client. A schedule will be provided for planning the effort as part of the detailed integration test plan. End-to-end testing is not limited to client back-office systems. It also includes transactions that start at the meter and include, but are not limited to, the applications. Each application team will provide resources to assist with planning and executing integration (system-to-system and end-to-end) and user acceptance testing. Operational Readiness Review activities are owned by client with inputs from Application and Testing Teams.

Appropriately-configured application and infrastructure environments will be available for the exclusive use of the Integration Testing Team. Data creation and management: The individual application team will provide the Integration Testing team with the data required to execute testing events (System-to-System Interface, End-to-End and User Acceptance). It is assumed that the data requirements for testing will be as well consolidated as possible, to minimize the amount of duplicate work needed from the Application teams.

Transaction Validation: The individual business process and application teams will make available resources to help with validation of transactions that are processed by the application as part of integration testing. Process Design Team will be available to support end-to-end testing based on the testing resource plan. Individual applications will be responsible for transfer of data (files, messages etc) across to other applications.

System to system integration testing of interfaces being built or modified to support the program will be the responsibility of the integration test team. End-to-End business transaction testing will be the responsibility of the integration testing team "End-to-end" refers to taking a data transaction from its initiating event until it is fully processed. Included are: Transactions that may initiate or terminate at meters, such as data reads or meter alarms and events, when client systems are part of the transaction processing [6]. Transactions that initiate or terminate at vendor systems, such as a Demand Response Event initiation that may be triggered in the Demand Response System and propagated to the MDMS via the Customer Information System. Testing of vendor systems (Ex: MDMS, ADCS, Meter Inventory etc.), will be conducted by the vendors and the test results will be presented to client. Security testing will be addressed in its own separate test plan. Testing of meters and the AMI network will be performed by the Meter and Network teams. Outside of the meter and network testing, the integration test team will execute test cases test that support business transactions defined during process workshops.

Testing of data transactions between the ADCS and meters or intermediate systems when client backend systems are not involved in the transactions will be tested by vendor. From an end-to-end perspective - the vendor systems will be delivered as a fully integrated solution and will be treated as a "black box". Testing of hardware, operating systems, middleware, and other infrastructure components will be performed by the individual application teams.

Operational Readiness Review will be performed by client, using the existing operational readiness review process. Testing of manual procedures that have been developed within client as part of the program will not be tested as part of end-to-end integration testing. Disaster / Recovery Testing – will be performed by client. Existing applications that currently have a DR plan may use the existing plan making edits if necessary. New application being introduced to client will have to develop a new Disaster / recovery plan.

Final Regression and Dry-Run testing will be performed if deemed necessary, by client with support from the integration test team [7]. Testing of existing interfaces developed by client- primarily legacy–to–legacy applications that have been altered by this project will be tested by the integration test team and will be included in end-to-end testing as well.

Service continuity testing will be performed by client if necessary, using existing and newly developed and documented procedures where applicable.

2.1.Testing Timelines

Each phase of testing will be executed for both releases within program, with the exception of Meter and Network Testing. Given the tight timeline schedules of the program, test phase will need to overlap with other phases to effectively utilize the overall testing duration. The figure below depicts how test phases may overlap as part of testing a release. The diagram is to be used as a reference of possible overlaps only and is not to be scaled

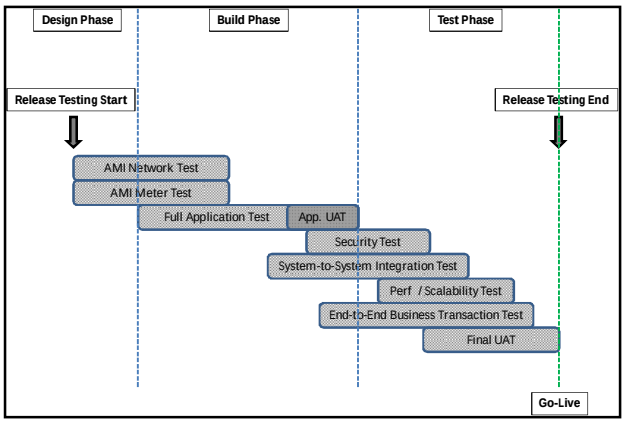

Figure 2: Release Testing Phase Timelines Structure

2.2 Organization Structure And Responsibilities

This section on the organization of the program testing team describes the hierarchy of the team structure. The responsibilities of the individual teams that make up the overall program testing team are also described in detail. The combination of the roles and responsibilities along with the testing approach convey how teams will work together to complete the execution of scenarios. The different teams will draw from their different areas of expertise in order to facilitate successful execution of all test cases.

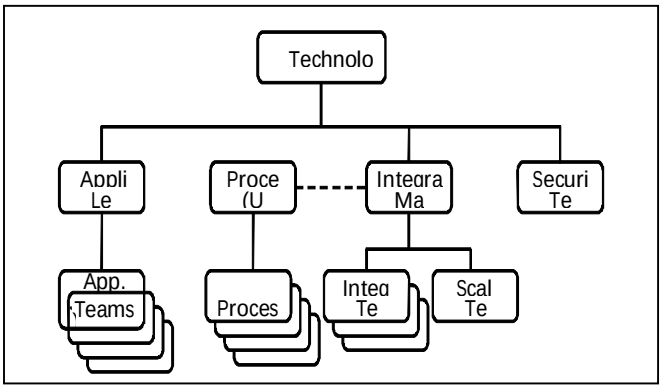

Figure 3: Program Testing Team Organization

3.Defect Management

The key measure related to software quality is defects. Defects discovered during testing need to be managed effectively to ensure software quality [3]. This task entails definition and implementation of a robust process that is communicated and followed.

The overview section on defect management mentioned four activities of defect management:

- Identifying a defect during formal testing
- Ensuring a difference between actual and expected results are true defects, rather than a bad test
- Ensuring defects are clearly documented, prioritized, and handed off to the right team to fix
- Monitoring that defects are fixed in a timely manner and that the fix is turned over to configuration control, are retested, and ultimately closed

3.1 Defect Tracking

Defects will be tracked using test tool.

3.1.1.Performed By

- Tester: Responsible for executing test scripts. Identifies and logs defects, provides supporting information, and validates the fix in the product once it is available.
- Developer: Analyzes and fixes defects; Performs unit testing and migrates code to source control.

- Business Process Analyst: Manages the defect across the testing effort. Reviews defects for completeness and validity. Meets with stakeholders to agree on the priority and action for defects. Assigns defects to developers for analysis and resolution. This person is also responsible for rejecting and deferring defects with <<CLIENT>> consent and producing a test closure memo at the end of test stage.
- Stakeholders: Meets with the business analyst to review defects, confirm priority and determine the action to be taken—placed in scope, deferred, or rejected. Typically includes the project manager, the test lead, and business representatives.
- Build Team: Builds fixes into the product and migrates the product to the testing environment for validation of the fixes.

3.1.2 Fields Captured For Each Defect

In this section of the test strategy, we now focus on three more important aspects of defect management – defect tracking, defect prioritization, and escalation of defects.

Status	Description
New	Defect has been detected and logged in Quality Center
Open	Defect has been assigned to the Developer / Business Analyst (in case of requirement defect)
Rejected	The defect logged is invalid. Developer / Business Analyst change the status to Rejected and assigns back to Tester.
Deferred	The defect is valid, but cannot be implemented in current release. Decision is made by Stakeholders to fix later. Assigned back to Tester.
Fixed	The defect is valid, and fixed in current build. Defect is then assigned back to Tester to re-test.
Closed	Final status that all defects need to be set to after going through the life cycle.

Table 1: Status Indicators of Defects

3.1.3 .Tracking Defects Through Lifecycle

The following process will be followed in moving a defect through its lifecycle, from when it is first detected until it is re-tested and closed [4].

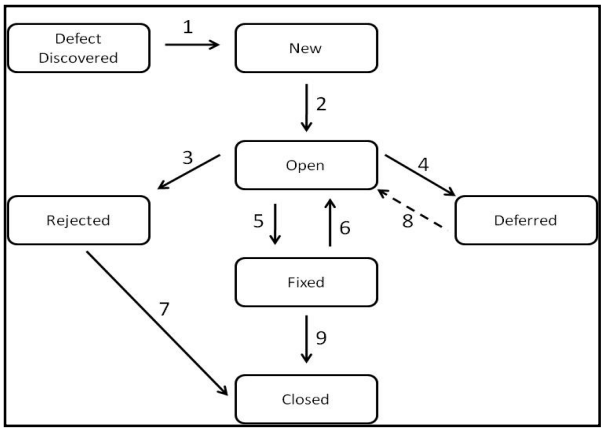

Figure 4: Defects shown through Lifecycle

3.2 Severity Levels

Every defect will be assigned a severity level to indicate its impact on the testing process and schedule. Severity levels defined are for the <<PROGRAM>> program and not for the individual applications of the <<PROGRAM>> program. However, the defects severity levels are consistent with the individual severity levels that are defined for the individual applications, from <<VENDORS>>.

Severity levels to be used are:

- Severity 1 – Critical (Fatal Severity / Catastrophic)

The application, component, or functionality under test is required functionality and there is no work-around available. Defect causes sudden or system wide disaster, operational outage or complete failure and occurs every time a test script is run. Examples are:

- A key transaction cannot be processed
- Defect hangs/crashes the program
- Defect makes it impossible to use the program
- Cannot perform major function
- Defect corrupts stored data
- Incorrect or missing data
- Defect conflicts with legal or regulatory requirements

- Defects that conflict with a high-priority (must-have) requirement
- Help files missing completely

Expected Response from Development Team:

Resolve Immediately. This halts testing in that area or module. This must be resolved to complete testing of the build and it must be closed or deferred before exiting the current test level. The expected turnaround for this defect is "ASAP" (a separate build may be required to fix this), or not more than 1 day.

- Severity 2 – High (High severity)

The application, component, or function under test is required functionality and a temporary work-around exists. Defect causes harm, productivity delays, impairs workflow and impacts multiple test scenarios. Examples are:

- A transaction can be processed, but only for certain input parameters
- There are issues causing intermittent crashes, loss of data, severe performance problems
- Defect seriously compromises the ease of use of the product
- Serious incorrect behavior
- Defect compromises corporate standards or security
- Defect conflicts with a medium priority requirement

Expected Response from Development Team:

Give High Attention. This requires a work-around before testing can continue in that area or module. The expected turnaround for this defect is <<X>> days.

- Severity 3 – Medium (medium / hindering)

The application, component, or function under test is required functionality and an acceptable work-around exists. Defect hampers, impedes, causes delay in the workflow, but does not significantly impact execution schedule. Examples are:

- A transaction can be processed with a logic error, but a work around exists
- Defect makes the software hard to use but would not prevent a customer from using the product
- Defect conflicts with a low priority requirement
- System does not report information that would be helpful to the user but otherwise the defect does not affect the functionality

- System temporarily displays incorrect information but does not mislead the user into causing damage
- Help topic not available

Expected Response from Development Team*:*

Resolve as part of the normal work queue. Testing continues to occur and the defect may be carried over to subsequent test levels. The expected turnaround for this defect is <<X>> days. If a defect is deferred, a workaround will be documented and delivered.

- Severity 4 – Low (Low / Annoying)

The application, component, or function under test is optional functionality and an acceptable work-around exists. Defect impacts a low number of test scenarios and execution can continue with minimal rescheduling. Examples are:

- Spelling error or a cosmetic issue on the GUI
- Unfriendly behavior that is annoying to the user

Expected Response from Development Team*:*

Testing continues to occur and the defect may be carried over to subsequent test levels. However, the expected turnaround for this defect is 1 week. If a defect is deferred, a workaround will be documented and delivered.

4.Result And Discussion

It shows an approach that provides assurance the release will enable the required integration interactions between the applications in order to facilitate and support "real-life" business transactions. This lifecycle starts with the application or system specific tests performed by application developers and infrastructure engineers (unit / string tests) and does not end until the business owners have given their approvals after user acceptance testing. The focus of integration testing is to verify that the applications that are linked together successfully enable "real-life" business transactions. As such, the scope of integration testing includes transactions that start at the Customer Information System and terminate at the MDMS or Legacy Interval Meter Data Management System. Integration testing is the point at which applications and their integrations are brought together for testing by independent testers. This type of testing is rigorous, comprehensive, and formal. These tests are started as soon as the applications are application tested, integrated and string tested. Tests are based on the business

requirements provided in the REQUIREMENT MATRIX and the transactions that are supported by the interfaces. End-to-End businesses transactions are tested as well during this test phase were scenarios that span multiple applications are executed and the integrated solution is validated so as to support real-life business transaction.

Our strategy covers the complete test lifecycle. This lifecycle starts with the first, informal tests done by application programmers (unit tests) and ends when the business owners have given their approvals after user acceptance testing. However, the strategy's focus is on integration testing, where all the applications are linked together to verify they can support real-life business transactions. As such, the scope of integration testing includes transactions that start or terminate at vendor systems or field-devices. User Acceptance Testing (UAT), where business owners and their representatives run a series of tests to independently verify that the systems will handle real world transaction. A sub-set of the end-to-end business transactions may be used for this testing phase. End-to-end testing is conducted when enough interfaces are delivered and validated to support an end-to-end business transaction. A test of a business transaction is meant to validate the applications can work together to support a real-life business process. Often these data transactions will start or terminate at either a meter or a vendor system. An example of a business transaction is the initiation of a DR event. This business transaction starts when the DR event is called in the Demand Response System and completes when the event information is propagated to the MDMS, DR Customer Notification System and Customer Information System.

5.Reference

1. ANSI/IEEE 829-1983 IEEE Standard for Software Test Documentation
2. Craig, Rick David; Stefan P. Jaskiel (2002). Systematic Software Testing. Artech House. p. 7.
3. Itkonen, Juha; Mika V. Mäntylä and Casper Lassenius (2007). "Defect Detection Efficiency: Test Case Based vs. Exploratory Testing"
4. Kolawa, Adam; Huizinga, Dorota (2007). “Automated Defect Prevention: Best Practices in Software Management”
5. Xie, Tao. "Towards a Framework for Differential Unit Testing of Object-Oriented Programs"
6. http://en.wikipedia.org/
7. http://www.guru99.com/