SUITE UPGRADE TESTING

PERFORMED BY

SOFTWARE AG SUITE INTEGRATION QA

Author: Rinku Hazarika

CONTENTS

1 Introduction

Upgrading is the process of replacing a product with a newer version of the same product to bring the system up to date or to improve its characteristics. Upgrade allows the customer to move to the latest and stable version of the product and leverage on the new and improved features.

Suite Integration QA team performs the upgrade testing of the entire suite. The validation covers all the supported paths and most of the supported platforms.

The document illustrates the comprehensive practices involved during the entire course of upgrade/migrate testing carried out by SIQA for different versions of products.

2 Types of Upgrade Testing in SIQA

2.1 Promotion Testing

2.1.1 Types of Verification

Upgrade testing is performed regularly as part of the promotion process. SIQA has been testing the upgrade since 9.5 release. The complete promotion testing is automated. The following are executed as part of the Upgrade Testing,

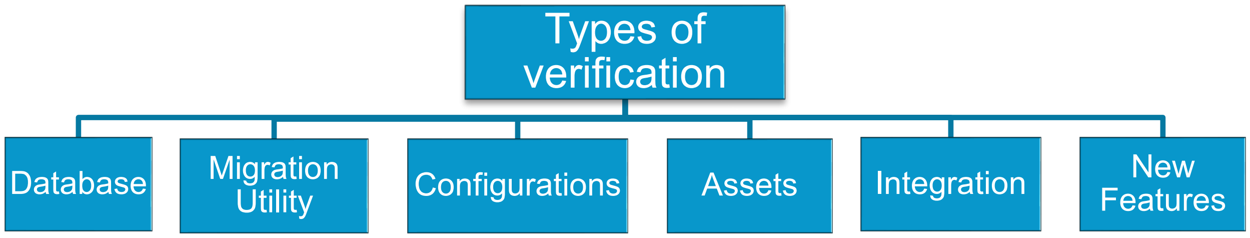


Figure 1: Types of Testing

- **Database**: Verifying data is not lost/corrupted after migration to the newer version.
- **Migration Utility**: Verifying the logs of migration utilities provided by the products and making sure that there is no error in the process of file system migration
- **Configurations**: Verifying configurations migrated to new version.

- **Assets**: Build assets in the source environment. Verifying that assets in the source are migrated completely and reliably to the new version.

- **Integration**: Building complex E2E Integration scenarios involving multiple products in the source environment. Verifying the scenarios in the newer version after migration

- **New Features**: Verifying new features of the newer version.

2.1.2 Supported paths

SIQA tests upgrade of all supported direct paths to the latest release. Refer to https://documentation.softwareag.com/ and search on “supported paths” in the left search box.

2.1.3 Environment

The following environments are upgraded and verified,

- Clustered Environment (MWS, IS and UM are in cluster)

- Product X = Other products

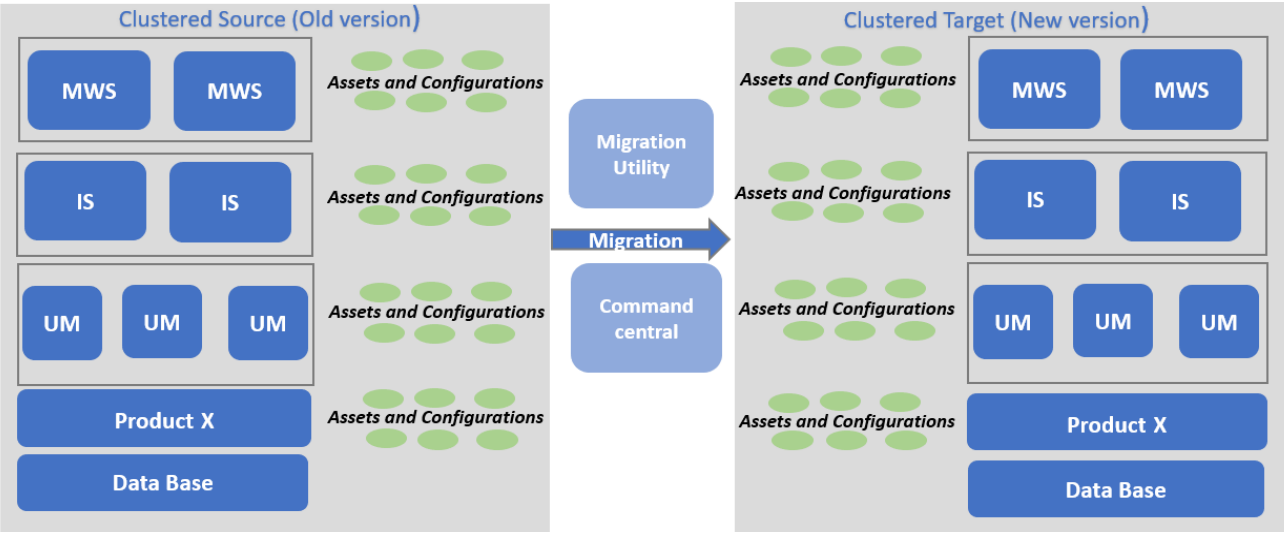

Clustered Source (Old version)
MWS
MWS
Assets and Configurations
IS
IS
Assets and Configurations
UM
UM
UM
Assets and Configurations
Product X
Assets and Configurations
Data Base
Migration Utility
Migration
Command central
Clustered Target (New version)
Assets and Configurations
MWS
MWS
Assets and Configurations
IS
IS
Assets and Configurations
UM
UM
UM
Assets and Configurations
Product X
Data Base

Figure 2: Clustered Migration

- Single Node Environment

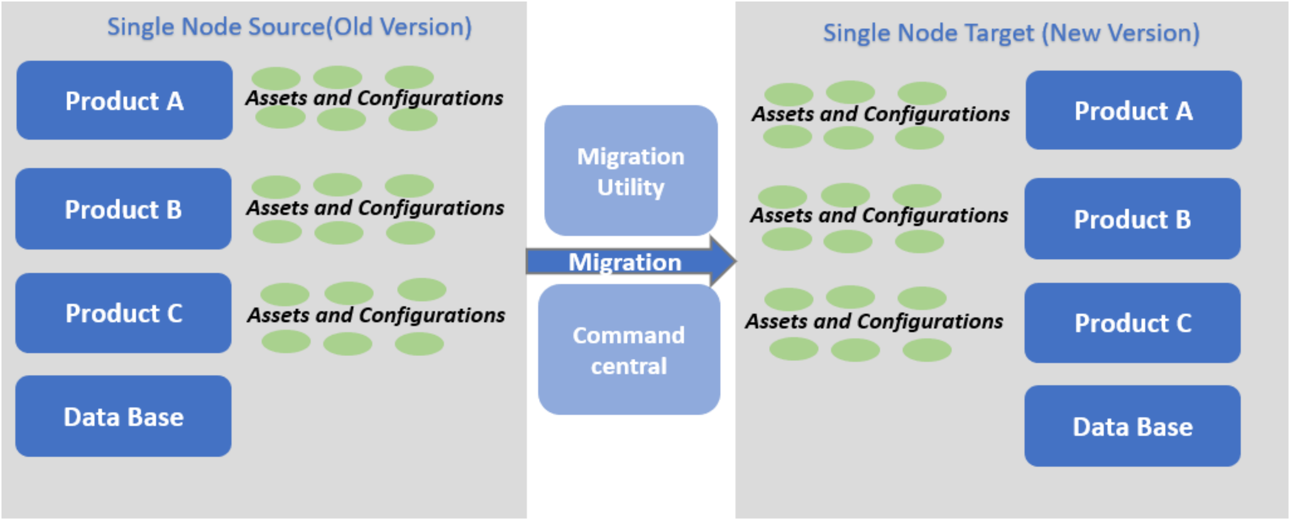

Single Node Source(Old Version)
Product A
Assets and Configurations
Product B
Assets and Configurations
Product C
Assets and Configurations
Data Base
Migration Utility
Migration
Command central
Single Node Target (New Version)
Assets and Configurations
Product A
Assets and Configurations
Product B
Assets and Configurations
Product C
Data Base

Figure 3: Single Node Migration

2.2 Platform Testing

SIQA tests the upgrade in different platforms. Automated test bench and Manual Testing is covered in platform testing. For AIX and Solaris, the migration is carried out manually and using Migration utility

Platforms covered are,

- Windows Server 2016,
- Linux RHEL7,
- AIX and
- Solaris

The below table summarizes the platform, environment and Database types tested in the SIQA Upgrade Testing suite for directly supported upgrade paths to the latest release,

Platform	Database Type	Migration Via:	Environment
RHEL7	Oracle	Migration Utility	Single Node
Windows	SQLServer	Migration Utility	Single Node
Windows	SLServer	Command Central	Single Node
Windows	Oracle	Migration Utility	Clustered
AIX	DB2		Single Node
Solaris	Oracle		Single Node
RHEL	Oracle	Command Central	Clustered

Table 1: SIQA Upgrade Test Coverage

2.3 Upgrade Guide Verification testing

SIQA has been actively involved in verifying the Upgrade Guide. Following two activities are carried out

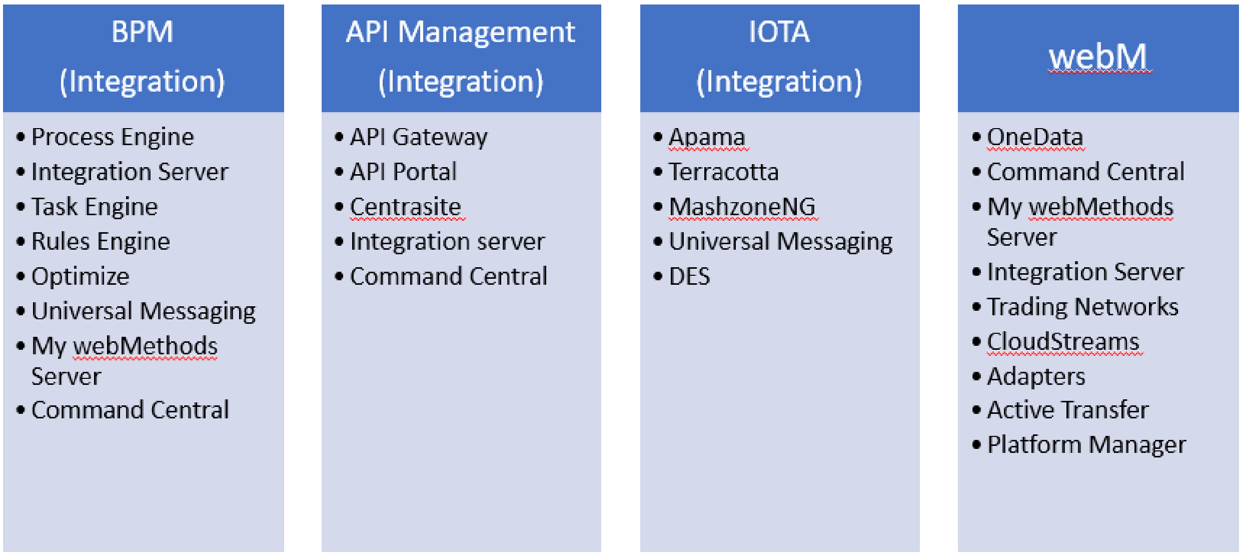
- Review of the Upgrade Guide for correctness of the contents
- Steps mentioned in the Upgrade guide are verified manually to identify any gaps in the documentation.

3 SIQA Upgrade Test Bed

3.1 Test Bed

3.1.1 Test Bed overview

The below diagram represents products covered in Upgrade Test Bed,



BPM (Integration)

- Process Engine
- Integration Server
- Task Engine
- Rules Engine
- Optimize
- Universal Messaging
- My webMethods Server
- Command Central

API Management (Integration)

- API Gateway
- API Portal
- Centrasite
- Integration server
- Command Central

IOTA (Integration)

- Apama
- Terracotta
- MashzoneNG
- Universal Messaging
- DES

webM

- OneData
- Command Central
- My webMethods Server
- Integration Server
- Trading Networks
- CloudStreams
- Adapters
- Active Transfer
- Platform Manager

Figure 4: Products Coverage

3.1.2 Source of Test Bed

The test bed for upgrade testing has been created considering all the aspects of Upgrade and products in the Installer Suite. The following are the source of the test bed,

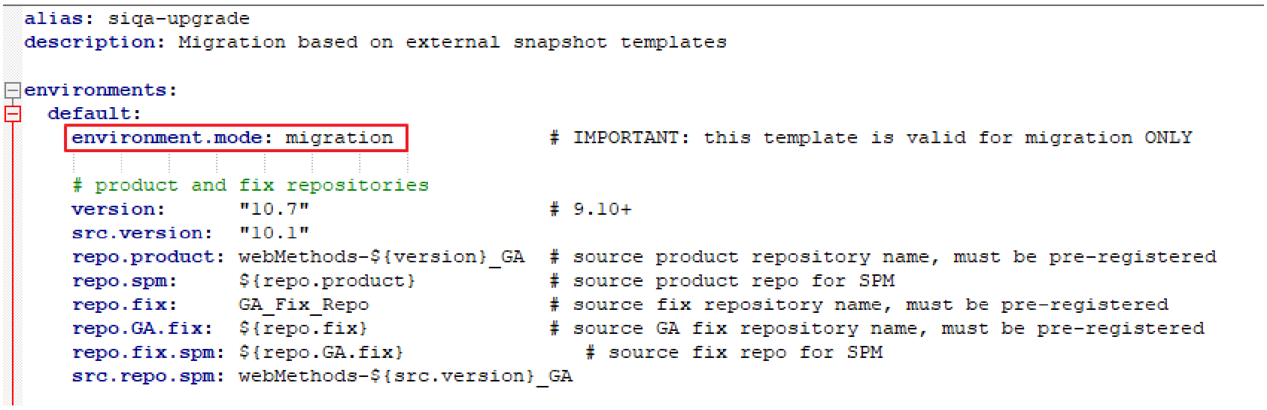
- Developer User cases for new features and functionality testing
- Customer issue triaging
- Interaction Matrix between Products
- Professional services
- Customer inputs

- Pre-Sales Demo
- Solution book
- Community Forums

3.2 Upgrade procedure

This section covers detailed upgrade procedure followed during our validation. The process is carried out in the following ways:

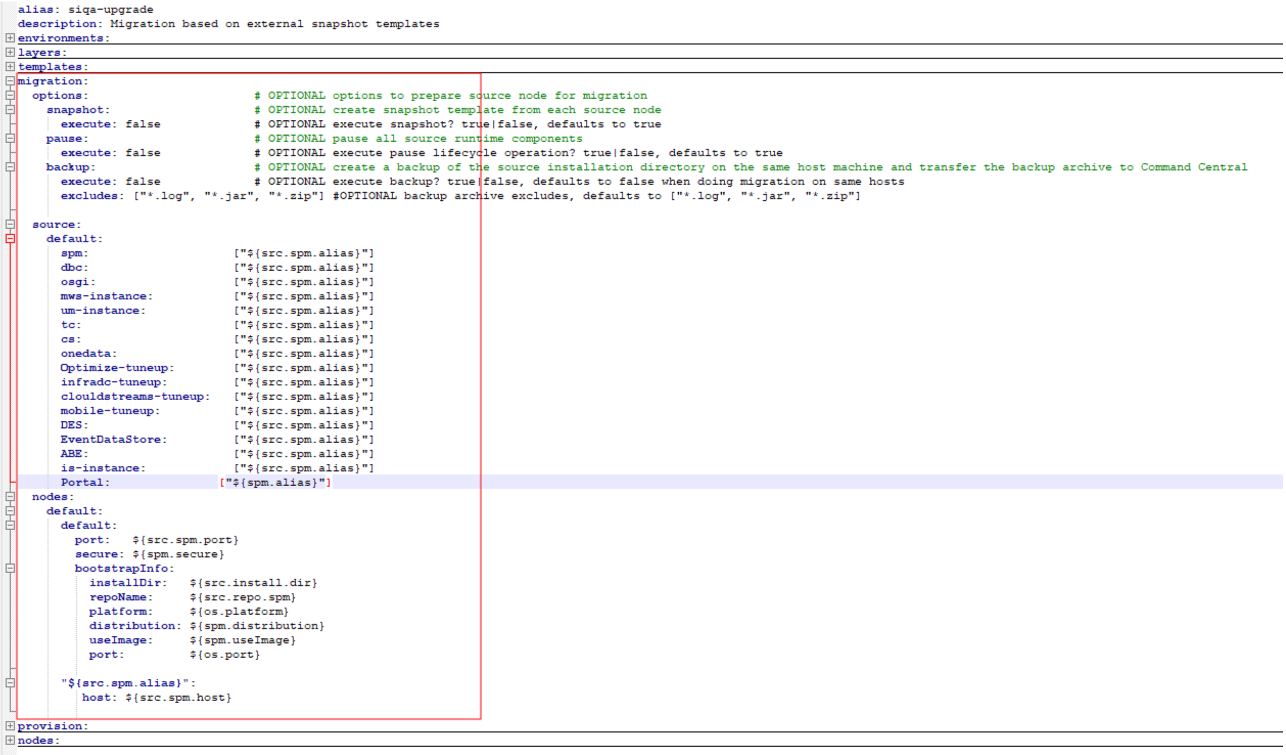
1. Using migration utilities and DB scripts
   a. Taking a backup of file system and DB.
   b. Installing the target version i.e. latest release.
   c. Data base migration: DB Migration Scripts are provided by the products. The target version can have additional tables and columns.
   d. File system migration using migration utilities: The products that support migration has an inbuilt migration utility that migrates the file system.
   e. Validation
      i. Migration logs
      ii. DCC Log
2. Using Command Central template
   a. Taking a backup of file system and DB
   b. Single click migration using command central template. Both Database and File system migration takes place in single click. The migration takes place in the order products are mentioned in the migration template. Below is a snippet of the migration template.



```
alias: siqa-upgrade
description: Migration based on external snapshot templates

environments:
  default:
    environment.mode: migration                   # IMPORTANT: this template is valid for migration ONLY

    # product and fix repositories
    version:       "10.7"                         # 9.10+
    src.version:   "10.1"
    repo.product: webMethods-${version}_GA    # source product repository name, must be pre-registered
    repo.spm:      ${repo.product}                # source product repo for SPM
    repo.fix:      GA_Fix_Repo                    # source fix repository name, must be pre-registered
    repo.GA.fix:  ${repo.fix}                     # source GA fix repository name, must be pre-registered
    repo.fix.spm: ${repo.GA.fix}                     # source fix repo for SPM
    src.repo.spm: webMethods-${src.version}_GA
```



```
alias: siqa-upgrade
description: Migration based on external snapshot templates
environments:
layers:
templates:
migration:
  options:                          # OPTIONAL options to prepare source node for migration
    snapshot:                       # OPTIONAL create snapshot template from each source node
      execute: false                # OPTIONAL execute snapshot? true|false, defaults to true
    pause:                          # OPTIONAL pause all source runtime components
      execute: false                # OPTIONAL execute pause lifecycle operation? true|false, defaults to true
    backup:                         # OPTIONAL create a backup of the source installation directory on the same host machine and transfer the backup archive to Command Central
      execute: false                # OPTIONAL execute backup? true|false, defaults to false when doing migration on same hosts
      excludes: ["*.log", "*.jar", "*.zip"] #OPTIONAL backup archive excludes, defaults to ["*.log", "*.jar", "*.zip"]

  source:
    default:
      spm:                        ["${src.spm.alias}"]
      dbc:                        ["${src.spm.alias}"]
      osgi:                       ["${src.spm.alias}"]
      mws-instance:               ["${src.spm.alias}"]
      um-instance:                ["${src.spm.alias}"]
      tc:                         ["${src.spm.alias}"]
      cs:                         ["${src.spm.alias}"]
      onedata:                    ["${src.spm.alias}"]
      Optimize-tuneup:            ["${src.spm.alias}"]
      infradc-tuneup:             ["${src.spm.alias}"]
      clouldstreams-tuneup:       ["${src.spm.alias}"]
      mobile-tuneup:              ["${src.spm.alias}"]
      DES:                        ["${src.spm.alias}"]
      EventDataStore:             ["${src.spm.alias}"]
      ABE:                        ["${src.spm.alias}"]
      is-instance:                ["${src.spm.alias}"]
      Portal:                   ["${spm.alias}"]
  nodes:
    default:
      default:
        port:   ${src.spm.port}
        secure: ${spm.secure}
        bootstrapInfo:
          installDir:   ${src.install.dir}
          repoName:     ${src.repo.spm}
          platform:     ${os.platform}
          distribution: ${spm.distribution}
          useImage:     ${spm.useImage}
          port:         ${os.port}

      "${src.spm.alias}":
         host: ${src.spm.host}

provision:
nodes:
```

c. Validation

   i. CCE log

   ii. Verify servers are up and running

3.2.1 Source Data

Data, Configuration and Asset information are collected from source and stored. This stored information is validated in target.

3.2.2 Executing Test Bed Scenarios

3.2.2.1 webM Product Testing; The following products are tested

- Basic Validation: Verifies whether the servers are up and running

Basic Validation (Servers up and running)
Integration Server
My webMethods Server
Universal Messaging
Analytic Engine
Infrastructure Data Collector
WebService Data Collector
API Gateway
API Portal
OneData
MashZone
Terracotta

- Integration Server: IS configurations and deployed packages in source are verified in target.

Integration Server
Common Directory Service
ISInternal, ISCoreAudit, Central User DB connectivity
Existing Scheduler Service
JMS connection Alias
JNDI connection Alias
Extended settings
MWS SAML resolver
Trigger Management
Primary ports
User Management
Caching
Logging
Running IS services (Custom and inbuilt)
Package Management
JDBC Adapters

- Cloud Streams: Verifies Cloud Stream package with the Sales Force Adapters.

Cloud Streams
Enabling Cloud Stream Connections
Provider connectors and connection names inside connectors
Update UserName and Password and Enable connection
Creating/Verifying/Deleting account in SalesForce
Import Cloud streams project from base release workspace and verify
Created a JDBC Connection Pool for CloudStreams

- My webMethods Server: Assets and configurations in source are verified in target.

My webMethods Server
User Management (Users/Roles/Groups)
Source Tasks.
Source Task Instances and its functionalities (Count/State/Attachments/Assignment/Events)
Creating new Task Instances
Directory Services
Date and time
CAF applications
Skins and Shells

- Process Engine: Business Process data in source are verified in target.

Process Engine
Configurations in WmPrt Package
Source Business Processes
Source Process Instances (Count/States)
Starting new Process Instances
Verifying state of the existing Source Process Instances

- Optimize: The optimize environment exported from source is imported in target and verified.

Optimize
Import Optimize Environment in MWS
Configure Imported Environment in MWS
Deploy Imported Environment in MWS
Configure IS and AE servers in MWS System Settings
Verify existing rule violation present
Create new process Instance to verify rule violation
Verifying Overview Configuration Count and Name, Business Data Count, EventMaps and KPIs
Check status in Process Analytics page
Check for Errors
New violation Rules
Around 15 itracs have been automated

- Infrastructure Data Collector: Functionalities and configurations are verified.

Infrastructure Data Collector
Discovering Assets IS
Discovering Assets MWS
Check the UM Relam KPIs in the Analytic Overview
Check the UM Interface KPIs in the Analytic Overview

- Rules Engine: Rules deployed in source are verified in target.

Rules Engine
Source Rules project in IS and MWS
Running a process model to trigger Rules
Verifying decision tables in MWS
Editing the decision tables in MWS and hot deploying it to IS
Editing the deployed Event Rules in MWS and hot deploying it to IS

- OneData: Functionalities and database configurations are verified.

OneData
Starting One Data
Source Users and Allocated Roles
Updating connections for PRD, STG
Verifying existing objects, column structure of Object, Constraint, mapping profile

- Command Central: Basic Command Central UI verification after migration.

Command Central
Command Central Migration from lower version to latest release.
Other supported products and DB migration using composite templates.
Verify running status of servers.
Verify filters in environments, instances, installations.
Access IS, MWS from CCE UI.
Monitor KPIs
Verify IS and MWS ports.
Adding keys, manifests, reports, repositories, environments etc to migrated CCE environment.
Validating instances, installations, keys, manifests, reports.
Starting and stopping of servers from CCE UI.

- CentraSite: Publishing assets from designer to Centrasite is verified.

CentraSite
Verify Roles for User
Add CentraSite Connection to Desginer
Publish project to centrasite from Designer

- Platform Manager: Verifies all products are present in SPM

Platform Manager
Login to SPM
Verifying installation of products (Deployer, Optimize Engine, Data Collector, Rules Engine, Process Engine)

- Active Transfer: Verifying the

Active Transfer
Source MFT Users and MFT Administrator Roles
MFT Ports
MFT Processing Rules
MFT Schedule Events
MFT IS Services

- Trading Network: Trading Network transactions executed in source and verified in target.

Trading Network
TN DB Connectivity
Source TN transaction
Initiating New TN transactions
Editing migrated document attribute, adding new document attribute
Source Processing rules
New Processing Rules
Business Monitoring

- Business Console: UI is verified.

Business Console
Source Business process in SBP
Source Process instances (Count and state)
Source Tasks and Task Instances (Count and state)
Verifying the SBP pages

3.2.2.2 Integration testing

- BPM Integration Scenarios: These scenarios cover integration between Integration Server, Process Engine, Task Engine, Rules Engine, Universal Messaging, My webMethod Severs, Optimize.

BPM Integration Scenarios
Simple process
Complex process with Joins, Gateways, Swim lanes, pools, error handler, time out, boundary events etc
JMS Triggered process by publishing a doc
JMS Triggered process with a service call
Process with tasks
Process with Rules
Task implementations like events, assignments, attachments, notifications
Sub Process, Sub Process with loops
Child Process
Callable process
Process instances and Task instance with all possible states (Cancelled/Error/Suspended/Expired/Continued/Completed)
Suspend and Resume Process
Resubmit the process
Stop the Process
Debug process and sub process
Logged fields and Errors
Part of Customer E2E Scenarios and Pre-Sales Demo

- API Management: API with all the policies and applications are created in Source gateway and published to Portal. These are verified in target by invoking APIs.

API Management
API Gateway
Global Analytics
REST API Analytics
SOAP API Analytics
ODATA API Analytics
WebSocket API Analytics
API Portal
Invoke and verify REST, SOAP, ODATA, WebSocket APIs.
Authentication Types (Basic, OAuth, JWT)
Operations (Get, Post etc)
Policies (Transport, Identify & Access, Request Processing, Routing, Traffic Monitoring, Response Processing) by invoking APIs from portal.
Applications by invoking APIs from portal.

- IOT: Integration Scenarios between Apama, Terracotta, MashZoneNG, Universal Messaging and Integration server are covered.

IOT Integration Scenarios
Create cache in Terracotta using Apama projects.
Send events from Apama to add and delete cache in Terracotta.
Lexmark a customer scenario.
JnJ a customer scenario
Json support in MashzoneNG
Apama Cluster Correlator
Apama, terracotta, MashzoneNg using static cache.
Accessing IS WSDL from Apama
Calling IS Flow Serivce returning XML content in MashZone NextGen Dashboard
Calling IS Flow Serivce returning CSV content in MashZone NextGen Dashboard
Apama_JDBC_MZNG (Add JDBC Driver and Data Source)

4 Customer Support

SIQA is actively involved in addressing customer issues.

- Testing in Customer Lab setup. Doing Upgrade in Customer Lab setup before customer, so that issues can be addressed beforehand.
- Answering to Upgrade Swat mails
- Supporting customer by trying to repro the setup in house.
- Verifying the upgrade fixes provided to the Customer.

5 Future Roadmap

- Include more cluster setups and failover scenarios.
- Having further discussions with PS, collecting their pain points and will try to include as part of our testing. (Currently on going).
- Cross platform migration (Only for supported products).

PRIME GUIDE

TESTING RECOMMENDATIONS

Performance

Author: Howard Pelling

Director, Expert Services - Performance

Version 0.1 | 25.07.2016

VERSION HISTORY (ONLY INTERNAL USE)

Version	Date	Author	Description
0.1	7/25/2016	Howard Pelling	First draft

TABLE OF CONTENTS

1 Introduction

The goal of testing is to ensure that the application or environment under test will behave as designed and in a predictable manner, through all phases of the Production life-cycle.

Functional testing is commonly performed on enterprise software, whereas other types of testing are often either not given the appropriate time and resources to execute them properly, or are totally neglected; a good example of the latter being testing restoring the environment after a backup.

Some types of testing, such as performance and security, require specialist knowledge, which is often not available within a small organization.

This document will describe many of the different test types and some of the recommendations involved in carrying out these tests.

2 Types of testing

2.1 Functional

The goal of functional testing is to ensure that the mechanical aspects of the application work as designed. It is usually organized and led by the QA team, in the QA environment, and will often employ end users to do the testing. The tests are generated from the ‘Function Specification’ documents and are designed to execute every path through the application.

Complete functional testing should not only test the ‘happy path’ through the application, but also the error and ‘edge’ cases; something that is often skipped.

The QA environment is often a scaled down version of the production environment, without high availability and in many cases on a lower class of machines. Otherwise, the architecture of the environment used to test the application should be functionally similar..

2.1.1 Recommendations

- Developers often use ‘Use Cases’ to create the application. Leverage these as the input for your functional tests.
- If there are no Use Cases, create them from the functional specifications. Use cases can be just a small snippet of functionality, or functions that span a complete activity. You can also join smaller Use Cases together to create a larger Use Case.
- Functional testing is not only used for UI driven applications; All applications should go through a functional testing phase.

2.2 Performance

Performance testing is intended to prove that the application, once in Production, will meet stated performance goals, ideally defined as Service Level Agreements (SLAs).

The Performance test environment must behave identically to the Production environment and thus is usually designed to be identical. In cases where the performance environment is not the same as Production, maybe due to budgetary constraints linked to the scale of the Production environment, special care must be taken to prove that the results of the performance tests are valid.

- Needs to be based on strictly defined SLAs, not vague generalizations
- End user network speed and latency may come into play
- Are there higher performance options available
    - Network interfaces
    - Caching
    - SSD
    - MWS and UM
- Effects of encryption on performance
    - If possible measure without

2.2.1 Recommendations

- Test on an identical system to that which will be deployed in Production. This includes the number of servers in a cluster, load balancers, database size, network interfaces, storage and many more factors.
- Anything but the simplest performance testing probably requires specialist knowledge. This knowledge includes the application, network, storage, database and testing

techniques, to name a few; oftentimes there is nobody on the team with complete knowledge of all the required aspects, so it is important to build the extended performance testing team from subject matter experts (SMEs) drawn from other parts of the organization.

- When a performance problem is found, break the application into small pieces and performance test each piece separately.
- Take note of whether the results are within a narrow band, and consistent, or whether there is a wide variation in values. Focus on making the variations as small as possible before moving on to the next snippet.
- The lowest values you can achieve from each snippet can be added together to give you the best case scenario.
- The worst case scenario can be approximated by summing the longest times.
- Read section 7, The importance of test data.
- Performance goals must be quantifiable and measurable.
- Understand the limitations of infrastructure elements, such as network and disk interfaces.
- Understand how any products use the infrastructure: for instance, a cluster of Universal Messaging servers share their state across the network, as does the Task Engine.
- Use strict change control to ensure that a record is kept of any changes. There is nothing worst that making ad-hoc modifications to the system, but getting worst results, then finding that you cannot reproduce the previous, better, results.
- Ensure that a record is kept of all system settings, application and product parameters. Log any changes made to these in the change log.
- Consider *all* factors when trying to track down a problem, for instance, is the use of encryption slowing the system.
- Tune the JVM before running any performance tests on the application.
- When testing a UI driven application, remember to take the fact that the user's client machine, and the network to that machine, may be sub-optimal into account.
- Build Test Cases from developer or QA Use Cases. As with Use Cases, multiple Test Cases can be joined to make a more complex test

2.3 Smoke

In the U.S., a Smoke test is a very simple test of passing a few transactions through the system and checking that everything works (i.e., no smoke appears).

In other areas of the world it sometimes means the same as a Stress test (load the system until it starts to smoke).

Smoke tests can be end to end, or just a small portion of the system; for example, a section of code that has been changed.

2.3.1 Recommendations

- These tests must use reasonably realistic data sets.

2.4 Soak

A soak test extends the time of normal tests to prove that the system will be stable over long periods of time. Sometimes a reasonably heavy load is applied for the duration, other more sophisticated tests will vary the load over the time period as may be expected in real life.

A soak test should be an end-to-end test of the whole application, including any maintenance that is normally performed while the system is running, such as live backups or data synchronization. If reports are run on the system at night, then include these runs in the test scenario.

Soak tests are not usually shorter than 4 hours, often 8 hours to simulate a work day, but can extend to a week or more.

What you are looking for in a soak test is a depletion of resources that either results in failure of all or part of the system, or a gradual decrease in performance. A decrease in performance may indicate database tables which are not correctly indexed, garbage collection thrashing or many other woes.

2.4.1 Recommendations

- It is important to include these tests as part of the schedule, with sufficient time to retest if the system fails.
- After the test, continue to monitor the system to see how long it takes for all the used resources to be released.
- It is best to test with normal volumes, or anticipated peak sustained volumes.
-

2.5 Stress

When stress testing you are measuring what happens when the system, or part of it, are subjected to extreme load. The goal is to take the system **to the point of failure**. By doing this we can identify the weakest links in the system.

When in Production, we can then monitor the weak links for signs that they are getting near to failure and take action to prevent it. If the point at which a component fails is close to normal operating volumes the information can be used to improve the performance of that component – be aware, once this is done it will move the failure point somewhere else.

An important part of a stress test is to find out how the system moves into failure. Is it gradual or catastrophic; and most importantly, is it predictable?

2.5.1 Recommendations

- Stress testing should be combined with, or driven by the volumes produced in Fail Over testing; e.g. in failing one node in a two node system, the load on the remaining node will double. It should be a joint IT/Business decision as to how many nodes may fail without affecting overall performance.
- Do not just put a normal heavy load on a system and call it stress testing.

2.6 Fail over

Fail over tests check the resilience of the system to full or partial failures of components. The scope can be as small as losing a single server out of a large cluster, to losing an entire data center. Fail over tests should anticipate different types of system failure, from systems which are still ‘alive’, but unable to process data, to systems that are totally unavailable. Ideally all possible combinations of system failure should be tested, but in a complex environment this is impossible. In these cases one should test the most likely combinations, possibly based on previous experience of real life failures.

An important element in failure testing is not just that the remaining systems continue to work, but that the load from the failed system(s) can be transferred to the working systems without impacting performance. In a multi-node cluster a decision must be made as to how many nodes are allowed to fail before performance is impacted. Conversely, tests can be executed to discover at what point a failing cluster begins to affect performance. This is useful when defining maintenance procedures, as it will also show how many nodes can be *intentionally* be taken out of service; for example, when doing a rolling upgrade.

2.6.1 Recommendations

- Define a minimum acceptable level of service.
- This may change depending upon business conditions (e.g. weekends vs busy times).
- Establish how many servers can fail, while still keeping within the level of service.
- Discover at what point a partial system will fail to keep up with transaction processing and start to back up, or lose transactions.
- Discover which elements have most impact if they fail.
- Discover which components lose, or buffer data when they fail without being able to transfer these data to a working node.
- Calculate, or establish through testing, the minimum number of nodes required to handle full load
- Make sure that people understand that when architecting for resilience, the servers under normal conditions will be *underutilized*.
- Do not forget that systems do not run well at 100% utilization. In a 2 node system average resource utilization should be at 45% or less on each server; so that if one node fails and the load is shifted to the remaining server the resource utilization will be less than 100%.
- It is not only CPU utilization that we have to worry about when transferring load from a failed component. It is very likely that all resources on the remaining servers will be impacted in some way; CPU, disk I/O, network bandwidth, threads, connection pools, to name a few.

2.7 Backup/restore/archive/purge

These elements, in practice, are rarely tested, although they are essential to the long term health of a system.

It is true that most organizations would consider backing up file systems and databases to be part of their normal regime, but how many times do they check that restoring these backups actually result in a working and consistent system?

Archiving and purging of data is also often required to ensure optimal performance. Exercising these functions to ensure that they not only work, but that they can complete within a defined maintenance widow is also important. All too often these functions are not attempted until so much data has accumulated that performance is severely impacted. By this time there may be so much data to archive/purge that the purging process cannot keep up with the data which is being added on a daily basis.

- Does a restore recreate a functional system?

- Use of restore for resetting test environment
- Can a backup be created in the available maintenance window
- How long does it take to do a restore
    - Are there any manual steps
    - What about inflight data
    - What about archiving
    - Crosses over into DevOps
- When does the volume of stored data start to affect performance

2.7.1 Recommendations

- Test both the backup and restore processes
- Test data archive/purge processes
- Use proper data
- Check whether additional, temporary indexes would help with archive/purge

2.8 Throughput

Throughput testing ensures that normal, expected, peak business loads can be processed comfortably by the system. Not all business cases have the requirement that all transactions are handled immediately, some systems are designed to intentionally buffer peak loads in order to spread the load out over a period of time.

- Note that throughput, soak and stress testing are different
- Projected volumes
- Limits

2.8.1 Recommendations

- Understand the expectations of the business, whether real-time, or load spreading is the norm. This will often hinge on SLAs the business has defined with its end users.
- Different types of peaks occur, usually following the business cycle. For instance, Fridays can be busy as people try to wrap up the incomplete work of that week; end of quarter and end of year may have different load profiles. Make sure that all valid profiles are considered.

- If possible, ensure that automated testing tools have the ability to adjust the load over the duration of a test to match the anticipated load profile.
- If maintenance procedures overlap with (hopefully light) business use, also check that the system will perform to requirements with these two functions running in parallel.

2.9 Security

Added layers of complexity, such as SSL, can negatively impact performance. They can also necessitate more round trips between different servers to perform the same functions as a system without these types of security.

2.9.1 Recommendations

- Testing a system without target security implementations in place can be considered, at best, a smoke test. All realistic testing, whether for performance, or functional reasons must be performed with the appropriate security in place.
- It is often necessary to involve people who understand how to implement and diagnose problems with security. Do not assume that the testing team is knowledgeable in the subject.
- Different security certificates may be required for the testing system, with the correct chain of trust.

2.10 Other

There are many other types of testing, of which the main ones that come to mind are intrusion, accessibility and HCI (Human Computer Interface).

2.10.1 Intrusion

For highly secure applications, such as in the finance and military realms, protecting data from unauthorized access is critical, but don’t discount this type of check for even basic applications. For the most sensitive systems 3rd party verification of the end to end environment may be required. Unless you have experience in this field it is better to let the experts handle it.

2.10.2 Accessibility

Accessibility deals with ensuring that the application can be used by people with impairments, or those who work in other languages. For some (especially government) applications, these are requirements which have to be met. For other applications it is a courtesy to your user base.

2.10.3 HCI

Users of your application may not always be local to your servers, running the latest and greatest hardware or software, or viewing the results on large screen monitors. If appropriate make sure that your testing is run using similar end systems to those that your end users have. Often there are many different devices; the use of tablets and even smartphones for accessing applications is becoming commonplace, alongside the more traditional monitor. Make sure that screen layouts and amount of data rendered is appropriate for all devices.

3 Scope of testing

As mentioned earlier, end to end testing should not be the first thing that you attempt. It is much better to start small and work up. Understanding the behavior of different modules will also enable you to understand where problems may lay in the larger system. For instance, if the best achievable time for a database call is 10 seconds when executed at the module level, it will not magically become 5 seconds when running an end to end test. It is possible to calculate best case end to end performance figures by taking the best case times from each module and summing them. The same applies to worst case numbers.

3.1.1 Recommendations

- As the scope increases, other expertise may be required to both run the tests and understand the results, such as database, or load balancer experts.
- Start small with modular testing, then combine modules into larger groups until you are eventually running end to end tests.
- Make sure that the most comprehensive tests include external factors, such as 3rd parties, or dependent systems provided by other parts of your organization. In the event that this is impossible, stubs must be introduced to the test harnesses that can faithfully reproduce the behavior of the external systems, and are also able to supply or consume the expected peak loads.

4 Environment requirements

Test environments and their complexity are very dependent upon the scope of the tests. It is common to execute modular and functional tests on scaled down environments, whereas end to end performance should be on a mirror of the production environment.

Always aim to run tests on an environment which is sufficient for, or better than, that required for that particular part of the test plan.

An environment to support comprehensive testing of a large system will be very expensive; it is likely that many reasons will be given why scaled down systems should suffice. If the proper systems cannot be acquired, it is essential that you are able to convey to the system owners what additional risks this introduces.

4.1.1 Recommendations

- Use the correct environment for the task at hand.
- Look for potential risks and communicate them
- Document the environment and configuration parameters in case the tests need to be repeated at a later date.
- In a brand new project consider using the 'real' production environment to run your performance tests.
- For end to end testing consider that the environment may extend outside of your local application realm.
- For high availability testing more than one data center may be involved.

5 Testing methodology

To be valuable, testing must be accurate, comprehensible and repeatable. There are many testing methodologies, just as there are many development methodologies, the important thing is to choose one and stick with it; otherwise comparing results from different test runs is very difficult, or impossible.

One of the main things that will help with this goal is automation, both in setting up the environment and running the tests. Ideally, installation of the environment is running one or more scripts, which do everything from installing the products, to configuring them. In this way you can ensure that if you have to rebuild the environment it can be easily reproduced. This may be important if after many changes you cannot replicate previous results. A product such as Command Central can definitely help. We have used CC to set up large, 24 node, clusters with almost the push of a single button.

As with the environment setup, performance testing should also be fully automated. In this way results from previous testing runs may be directly compared.

Tools are available for automating functional and other types of testing, although the cost benefit ratio may not be as high.

It is important to understand, and have agreement on, the scope of influence you have on the overall project. Project go-live dates are usually set without regard for whether the testing is successful or not. What would happen if you cannot prove that the system can meet the required SLAs before the go live date? What if you *can* prove that the system does *not* meet the SLAs? Are you able to stop the system going into production, or is your input only advisory? These are decisions that are best made before testing is started.

5.1.1 Recommendations

- Automate
- Automate
- Automate
- Make detailed change control an integral part of your process
- Test reports should contain detailed specifications of both hardware and software configuration for that specific test.
- Archive the environment setup and test scripts in case they are needed later.

6 How to measure

Metrics are the cornerstone of any testing process. If the correct metrics are not available it may be impossible to prove your test case. The test case itself should define the target goal and the metrics required to prove compliance to this goal. A common mistake is to base your testing on metrics that you have, rather than the metrics that you need.

There is a balance to be made when collecting metrics. Often collection of the metrics can affect the results themselves. Consider a profiling tool. When collecting the data the tool has to interrogate many areas of the code, which all takes CPU cycles, which on a heavily loaded CPU will make fewer cycles available to do useful work, thus slowing down the application.

With this in mind, care should be taken to not ‘over instrument’ an application; this includes setting the logging level too high.

Don’t forget that diagnostics and performance measurements are two different things, just as rehearsal and performance are for a musician. When diagnosing problems you want the maximum amount of information that will allow you to identify the cause, at the expense of affecting the performance. However, when performance testing you should keep the metrics as lightweight as possible. Keep in mind that the metrics that you choose for performance testing should also be part of the production monitoring metrics, as these will show when production is under stress.

6.1 Metrics

The actual metrics you collect will change dependent on what you are trying to prove. You may need metrics that are not available by default and you will have to either find a way to instrument the system to provide these, or use existing metrics in combination to generate them.

Don't forget that many metrics you will need are not all SAG product based. The operating system, storage systems, databases and many other external sources may need to be part of your dataset.

Be careful that you understand both what the metric is telling you, as well as any possible limitations with the collected data. For instance, CPU load statistics in a VM may only be showing you the resources used by that VM. Your application may only be using 20% of CPU, whereas another VM may be using 80% of the raw hardware power. The reason that your CPU is showing 20% utilization is that it *can't get* any more CPU cycles, not because it is lightly loaded. Monitoring the physical hardware that hosts the VMs can show you things like this.

When collecting metrics there are 3 values you should be interested in, maximum, minimum and mean. As described earlier, when measuring response times, the min and max will help you establish best and worst possible times; but they will also show you the amount of variability in your responses. The goal should be to reduce the amount of variance as much as possible.

In some cases neither the product, nor external tools can provide a needed metric. In this case one of your only options may be to insert some code in your application that will write an entry to the logs.

Log files can also be useful in other cases. MWS writes a timestamp in the log for each http request, along with the request details. These can be used to make ad-hoc measurements across different functions by subtracting the timestamps. When using timestamps be careful when comparing across servers, as the timestamps are probably not synchronized.

6.1.1 Recommendations

- Do not rely on one test run for your results. Make several runs and evaluate your entire result set
- If an individual set of test run results feel wrong, they probably are wrong. Explore why.
- Repeatability is key
- More complex test scenarios are more difficult to understand. Simple is often better.
- Your job in testing is to prove that you cannot make the system fail the test, not to prove that it can pass the test. Don't look the other way if you see a potential problem; it will be the first one they find when you put the system into production.
- Ensure that you actually understand what the test results are telling you.
- Testing is an iterative task; test, tune/fix, repeat.
- Change one thing at a time.
- Use change management. Nothing is worse than not being able to reproduce that stellar result at a later date.

- Use modular optimization before attempting to optimize a complex system

6.2 Tools

There is no 'one tool' that will collect all the metrics you need. Some products are better than others at providing information; for instance, MWS makes a huge amount of information available through its JMX interface. Integrating collection tools with automated testing systems can also sometimes be a challenge.

Testing tools such as Load Runner and Silk Performer can be found at many customers, but they are expensive for very large deployments. In these cases a combination of different tools can be used. In one engagement, where we were testing response times for 15,000 end users, we used 3,000 highly instrumented users for the main metrics, 10,000 cheaper licenses for a broad brush monitoring approach and 2,000 users driven through JMeter just to add the required load. This approach worked well.

When using tools to create load it is important to also monitor the machines which are creating the load to ensure that you are not running out of resources there, and consequently artificially throttling the potential peak load.

In addition to tools to drive load, you will need tools to collect the metrics. These will change, dependent upon your test cases, but include:

- vmstat
- top
- iostat
- Windows performance tools
- Virtual machine monitoring tools
- Database monitoring tools
- Tools internal to the SAG product stack
- … and many more

7 The importance of test data

Without proper test data, test results can be considered dubious at best.

Ideal test data would match the volume, diversity, content and error rate of production data. We will take a look at each of these aspects.

7.1 Volume

Volume is particularly important when conducting performance tests. Resubmitting data part way through a test can cause databases, or browsers to use cached information, which will be accessed faster than if the data has to be read from other data stores. For long running tests this can be a challenge.
If possible use redacted production data, i.e. production data with any ‘sensitive’ fields, such as account numbers, or government issued ID numbers, replaced with a random string; while taking into account the next section on ‘Diversity’ of data.

7.2 Diversity

7.3 Content

7.4 Error rate

  - Unrealistic data proves nothing
  - Base data on type of testing
    - Performance testing should use a typical mix of data as would be seen in a production environment
    - Stress testing may use worst case data
      - Large messages
  - Include erroneous data
    - In real life proportions
- Steps for establishing and executing a test plan
  - What type of testing
  - End goals or SLAs
  - Audience for results
  - Environment required
  - Expertise required
  - Test cases
  - Metrics to prove test cases
  - Data required
  - Tools required
  - Test harness
  - Run tests
  - Sanity check results

8 Conclusions

Testing covers many different aspects of the business system life-cycle.

The reason that we test is to reduce business risk. Identifying these risks allows us to reduce as many of them as is practically possible. But eliminating all risks is not possible, so it is important, at the conclusion of any type of testing, to flag the remaining known risks to the system owners, so that they are aware of which risks remain.

ABOUT SOFTWARE AG

The digital transformation is changing enterprise IT landscapes from inflexible application silos to modern software platform-driven IT architectures which deliver the openness, speed and agility needed to enable the digital real-time enterprise. Software AG offers the first end-to-end Digital Business Platform, based on open standards, with integration, process management, in-memory data, adaptive application development, real-time analytics and enterprise architecture management as core building blocks. The modular platform allows users to develop the next generation of application systems to build their digital future, today. With over 45 years of customer-centric innovation, Software AG is ranked as a leader in many innovative and digital technology categories. Learn more at www.SoftwareAG.com.

software AG