

INFORMATION NOTE 005 - SYSTEMS TESTING REQUIREMENTS AND ACTUARIAL INVOLVEMENT

Classification

This document is classified as an Information Note.

Abstract

This Information Note outlines the system testing required when software is upgraded, or new software is implemented.

Purpose

This Information Note provides an overview of the testing process and the steps that should be covered when an actuary is included in a testing process.

Legislation or Authority

Actuarial Society of South Africa Systems and Technology Committee.

Application

This Information Note is aimed at:

- Actuaries or actuarial students involved with product development;
- Actuaries or actuarial students employed or contracted by a company which is changing administration systems; and
- Actuaries or actuarial students asked to sign off on migration or system related issues.

Author

Actuarial Society of South Africa Systems and Technology Committee

Status

Version 1.0 Effective from February 2021

1. Background

The increasing complexities of insurance and similar products and business processes have led to an increased dependence on technology and IT systems. This in turn has led to the development of various Administration Systems (AS) which enable institutions and service providers to manage the end-to-end life cycle of financial products. AS should allow for the recording of the relevant personal information and risk characteristics of the policyholder or member and validate this information as it is being captured.

The software demands on AS are enormous, driven as they are by, amongst others, the requirements for various portals for the insurance carriers, agents, policyholders, potential customers, and customer relationship executives. While AS is complex, the requirements of underwriting profitability, experience analysis and data extracts make it even more challenging for an organisation.

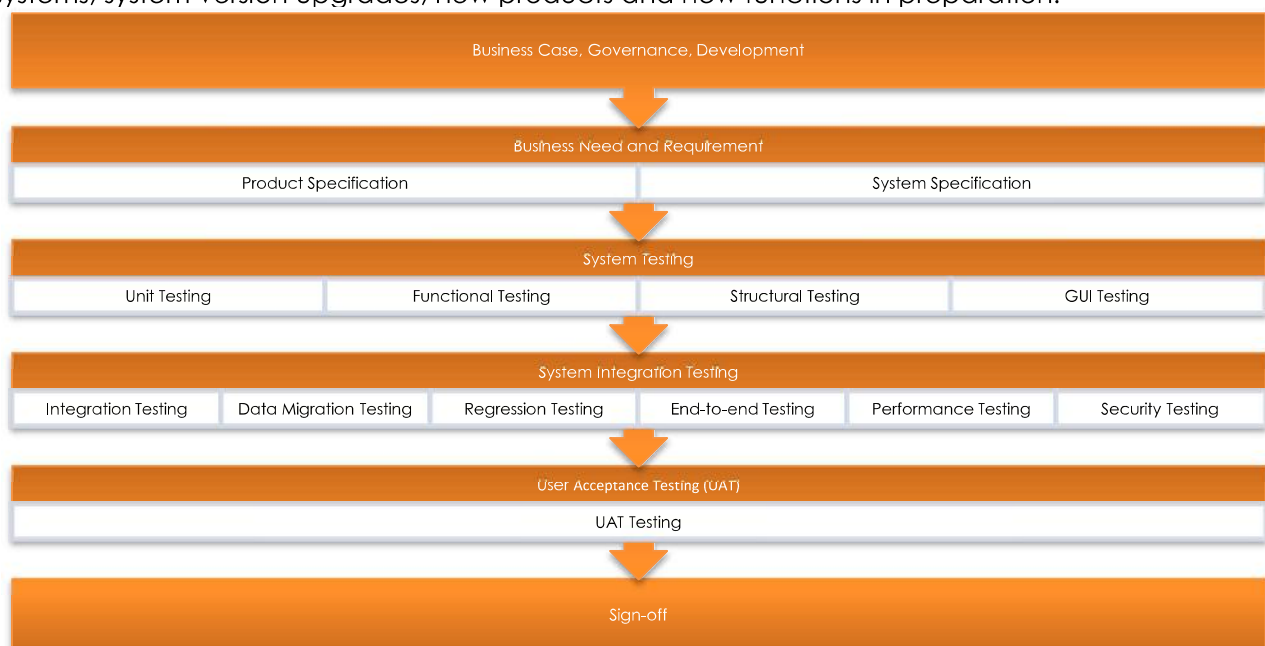
The impact of system defects therefore represents a significant risk to insurers, retirement fund administrators, medical schemes, banks and other risk-taking entities. These defects can have a negative impact in terms of flawed transactions, reduced customer base, miscommunicated policy information and compromised data integrity.

This Information Note was prepared because actuaries are being included to some degree in the software testing process. It makes sense to include actuaries in the process because they understand the full value chain of the software and are its end users.

Software is always configured for each client and this means that testing should always be conducted before the software is used.

2. Policy Administration System Testing Hierarchy

As with almost any technical process, the Policy Administration System testing process has a prescribed order in which steps should be completed. The list shows Policy Administration System testing categories, arranged in chronological order. These are the steps required to fully test new systems, system version upgrades, new products and new functions in preparation.



There are many types of software testing, but this document focuses on those that a risk-taking entity would employ. Each of the testing categories is described in the next few sections of this document.

3. Specifications

Before system development can begin, the business requirement, such as a product specification, should be in place and all the information should be translated to a system technical design or system specification. The system development and configuration costs are normally determined by the specifications. Any changes to the specifications after development has commenced may result in additional costs to the company.

Testing and development can only be as good as the specifications. When specifications are unclear or contradictory, development errors or incorrect testing outcomes will result.

4. Software Testing

Testing is inherently a risk-based activity. Most companies would not want to add testing to the cost of the project if there was no cost associated with the risk of failure. In other words, they would want to pay for testing if they are aware of the costs associated with a failed system implementation. Exhaustive testing is impractical for most applications under development, but crucial where the success or failure thereof has significant financial implications.

Although testing is a process, it is very much a dynamic one in that the process will change somewhat with each application being tested. There are several variables that affect the testing process, including the development process itself, software or system risk, customer/user participation, the tester's skillset, use of tools, testing budget and resource constraints, management support, and morale and motivation of the testers.

4.1. Scope of Testing

The scope of the required testing drives the testing process. A narrow scope may be limited to determining whether or not the software specifications were correctly implemented. The scope broadens as more responsibilities are assigned to software testers.

The scope of software testing could be expanded by adding the following responsibilities:

1. Software testing can compensate for the fact that the software development process does not identify the true needs of the user, and thus can be used to test whether or not the user's needs have been met regardless of the specifications;
2. Testing for defects early in the software development process when they can be corrected at significantly less cost than if they were detected later in the software development process;
3. Removing defects of all types prior to software going into production – this is significantly cheaper than trying to do this once it has been implemented; and
4. Identifying weaknesses in the software development process so that these processes can be continuously improved, resulting in better future software. Mature processes that incorporate these lessons will produce software that operates more effectively and efficiently.

4.2. Test Cases

A Test Case is an action executed to verify a particular feature or functionality of the software application. Test Scenarios, which are represented by a set of actions, can be rather vague or cover a wide range of possibilities. Testing should be very specific – this creates the need for Test Cases.

While drafting a Test Case, the following information should be included:

- A description of the feature/functionality to be tested;
- An explanation of how this feature/functionality will be tested;
- Version of the system, database, and name of the testing environment;
- Inputs or actions and expected results; and
- Any documentation or attachments that support the expected results.

A Test Case should be set up such that the number of steps a tester needs to make is kept to a minimum. For example, rather than testing the entire claims functionality in one Test Case, it could be split into the following Test Cases: Capture the claim; Authorise the claim; and Pay the claim.

4.2.1. Designing Test Cases

Test Cases should be designed to cover all the functional scenarios of the software in real-world usage. They should represent likely business and policyholder actions and objectives. If the Test Case scenarios are considered very unlikely to happen in the real world, testing them may not be rated as high priority; however, extreme or low frequency possibilities should not be excluded from the Test Cases.

Test Cases are designed in a simple language and manner to make the testing process more straightforward. As a best practice guideline, it is recommended that, in conjunction with industry experts, an actuary and business analyst review the test cases before they are finalised.

4.2.2. Test Pack and Coverage

A test pack consolidates major business scenarios and can be grouped by functionality or product. The test pack can then be reused for testing every patch release and system version upgrade. It should be reviewed and updated if the functionalities or product structures have been changed or when a new product is added.

A subset of a test pack can be executed depending on the change in the system. The same test pack can also be used for the development of an automation tool at a later stage.

4.2.3. Best Practice for Writing a Good Test Case Example

- Test Cases need to be simple and transparent;
- Test Cases must be created with the end user in mind;
- Test Case repetition should be avoided;
- Functionality and features of the software application should not be assumed whilst preparing Test Cases;
- 100% coverage should be ensured;
- Test Cases must have a unique identifier;
- Test Cases must be repeatable and capable of standalone use;
- Test Cases must be subject to peer review; and

- Test Cases should be prioritised.

4.3. In-development System Testing

The testing process, which includes four stages as listed below, should then begin during the construction phase of development.

4.3.1. Unit Testing

Unit testing is a method of validating the smallest components of the system, ensuring that they can handle known inputs and outputs as intended. This form of testing should be carried out by the developer whilst each component is being developed. Unit testing is normally conducted by the programmer who writes the code, whereas successful development teams will often have a peer perform the unit testing.

Unit tests are usually expressed in terms of the coverage they provide. The coverage is as follows:

- Function coverage: each function executed by at least one Test Case;
- Statement coverage: each line of code is covered by at least one Test Case; and
- Path coverage: every possible path through the code is covered by at least one Test Case.

4.3.2. Functional Testing

Functional testing addresses concerns surrounding the correct implementation of functional requirements. Commonly referred to as 'black box testing', this type of testing requires no prior knowledge of the underlying implementation. Functional testing is not concerned with how processing occurs, but rather, with the results of that processing.

Functional testing mainly focuses on:

- Mainline functions
- Basic usability
- Accessibility
- Error conditions

To test the functionality of a system, the following steps must be observed:

- Identification of the test input (test data);
- Computation of the expected outcomes with the selected test input values;
- Execution of Test Cases; and
- Comparison of actual and expected results.

Factors to be considered in functional testing:

- Product rules
- Business rules
- Validations

4.3.3. Structural Testing

Structural testing is sometimes referred to as 'white box testing' because knowledge of the internal logic of the system is used to develop test cases. A test team would verify a block

of code by reading the code to determine if the code would work reasonably. They then run an application to structurally validate the code. The advantage of this approach is that code can be tested, which would not be the case if only functional testing was performed.

The following are white box testing techniques:

- Statement Coverage: executing all statements at least once;
- Decision Coverage: executing each decision direction at least once;
- Condition Coverage: executing each decision with all possible outcomes at least once;
- Decision/Condition Coverage: executing all possible combinations of condition outcomes in each decision, with all iterations treated as two-way conditions exercising the loop zero times and one time; and
- Multiple Condition Coverage: invoking each point of entry at least once.

4.3.4. Graphical User Interface Testing

There are two types of User Interfaces (UI) for a computer application:

- Command Line Interface, where the user types a text command and the computer responds to that command; and
- Graphical User Interface (GUI), where the user interacts with the computer using images rather than text.

GUI testing involves checking the screens and input controls such as menus, buttons, icons, toolbars, menu bars, dialog boxes, windows, etc.

UX (User Experience) testing should also be conducted, where the GUI is assessed for efficiency, intuitiveness, processing times, user friendliness, etc.

4.4. System Integration Testing

System Integration Testing involves the overall testing of the complete system, together with many subsystem components or elements; it is part of the life cycle for collaborative projects.

System Integration Testing includes the six stages outlined below:

1. Integration Testing is done after the completion of System Testing. It refers to the tests performed to ensure the correct interactions between integrated components of the system code and other systems.
2. Data Migration may be required following the introduction of a new system, when data and history are moved from one system to another. This may involve a system migration or consolidation in which one or more legacy systems (i.e. old system or previous systems that are no longer being upgraded or supported) are replaced, or following the deployment of an additional system that will sit alongside the existing system.

Testing should differentiate between Pre-Data Migration Testing and Post-Data Migration Testing.

The role of the actuary in Migration Testing can be detailed as follows:

- The actuary should be involved in the scope verification, i.e. determining which data should be included or excluded, and considering whether it complies with the latest legislation;
 - The actuary should review and approve the consolidated, reproduced or updated product specifications, which may be prepared by an IT person with knowledge of the legacy system;
 - The actuary should approve the product mapping and ensure that it complies with TCF requirements (product mapping is where components (benefits, fees, lives assureds, etc.) from the previous system are mapped to similar components on the new system);
 - The actuary should test the valuation data from the new system and compare this with the results from the legacy system to ensure that all information has been carried over correctly; and
 - The actuary should sign off the migrated data.
3. Regression Testing is retesting unchanged segments of the system. It normally involves re-running tests that have already been executed to ensure that the same results can be achieved currently as were achieved when the segment was last tested. This would typically be carried out following a patch release or system upgrade, for example.
 4. End-to-End Testing is a technique used to test whether the flow of an application from start to finish performs as expected. It is used to test the flow of a new policy being captured all the way through to the processing of enhancements and claims, to approval and closing of the policy.
 5. Performance Testing, also called Load Testing, is the process of determining the speed or efficiency of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions.
 6. Security testing is for the protection of confidential information. Security assures third parties that the data is safe. The amount of security provided will be dependent upon the risks associated with compromised or the loss of information or associated monetary loss; which the actuary should be made aware of and sign off from a business perspective.

4.5. User Acceptance Testing (UAT)

User Acceptance Testing (UAT), also known as Beta-testing, is the process of verifying that a Policy Administration System works correctly for the user. It is the final phase of the system testing process. At this stage, actual users are drafted in to test the software. UAT is important because it helps to demonstrate whether the required business functions are operating in a manner suited to real-world circumstances and usage.

4.5.1. What should be done before UAT testing can start?

Before UAT testing can start, there are a few criteria that must be met:

- Business requirements must be developed and signed off;
- Application code should be completed and developed;
- System Testing and System Integration Testing should be completed; and
- UAT Environment should be ready.

4.5.2. Stages of UAT testing

A set of criteria to verify whether the system is working needs to be created and assembled. UAT Test Cases must also be created and tests must be run and recorded. Finally, an orderly sign-off must be completed.

5. Defect Life Cycle

A defect is an undesirable state in a system which can be due to failed processes or procedures. Testers need to be able to identify two types of defect:

- Variance from product specifications: a defect from the perspective of the builder of the product; and
- Variance from what is desired – a defect from a user (customer) perspective.

Typical system defects can occur when:

- A developer improperly interprets requirements i.e. they misinterpret what the user wants, but correctly implement what they believe is required;
- Users specify the wrong requirements, or specify requirements such that there is room for interpretation;
- Design specifications are incorrect - that is, the system designer does not achieve the system requirements but the design as specified is implemented correctly;
- Errors in coding;
- Data entry errors;
- Testing errors; and
- Mistakes or omissions in error correction.

What is the Defect Life Cycle?

The Defect Life Cycle or Bug Life Cycle is the specific set of states that a defect goes through, from discovery to the defect having been fixed to re-running the Test Case which initially highlighted the defect.

6. Signing off the testing

A system Go-Live date is usually agreed once all parties are satisfied with the system. From an actuarial point, signing off is required on migration, defects and system rules.

The Go-Live process might exclude certain products or functionalities. It is important that these are well documented and understood before the Go-Live date is agreed on. A system vendor fee could be determined by the Go-Live date and all parties should be aware of this, and whether the Go-Live date is realistic. Data or system repairs are much more difficult and costly once in production.

7. Importance of Proper and Effective System Testing

Risk-taking entities, like insurance companies, rely heavily on the administration systems used to run their businesses. It is therefore essential that these systems should operate properly, with all functionality working as required by the business.

The benefits of proper system testing can be summarised as follows:

- Minimises the number of system issues found after the system is put into production;
- Reduces the cost of liabilities caused by system errors and incorrect calculations;
- Ensures that the system is working as expected for all products provided;

- Ensures the correct interaction between integrated components of the system code and other systems; and
- Ensures that the information provided to policyholders corresponds to the system functionality.

Effective system testing:

- Reduces the costs of testing and redevelopment;
- Reduces timeframe required for testing;
- Minimises the time needed for the implementation of the system; and
- Can include end users in the testing process.

A balance will need to be found between effective testing and conducting as much testing as possible. Ineffective testing can occur when:

- Insufficient and non-automated unit testing is conducted before the system goes into production (this can place a major burden on the system once it is in production);
- There is perceived flexibility in testing timelines and duration, reducing testing coverage; and
- Minimal code reviewing takes place, thereby allowing for poor coding practices to pass unnoticed.

Ineffective testing could increase the time and cost of testing if issues are not found timeously.

The following sections discuss automation testing, which can contribute to making system testing more efficient.

8. Automation Testing

Automation testing refers to the use of software (usually separate from that being tested), specifically created to control the execution of tests and the comparison of actual outcomes with the outcomes expected. This type of testing is useful following code fixing, patch releases and system version upgrades. Automation testing can be used at any stage of the testing, as long as the testing environment is stable.

Automation testing can take over repetitive and tedious tasks, where humans tend to lose concentration and overlook things errors. Machines are more reliable and faster, thereby enhancing production efficiency.

8.1. Advantages of Automation Testing

- Although there is a significant initial investment for developing an automation tool internally (as opposed to adopting one from the market), it has been shown that the return on the investment is a long-term one – this approach will save time and ultimately costs.
- Automated testing is faster and can ultimately be much cheaper than human testers;
- Automated testing does not require a lot of people, needing only an automation engineer to write the scripts to automate the test cases, instead of a larger group repeatedly conducting manual testing.
- The scripts can be reused if the version of the insurance system changes, without changing the basic structure.
- The same tests can be carried out repeatedly, without the risk of steps being omitted.
- Automated testing can be executed at any time. It can be run over night when there is a low workload on the server – this allows the tester/test automation engineer to see the results in the morning.

- Automated testing creates an agile development environment that is not possible with human testers, giving the company the ability to jump the “coding phase/ testing phase” paradigm.

8.2. Limitations of Automation Testing

- Initial development of automation software can be very costly if in-house development is required.
- The automation engineer should have both programming skills, adequate insurance background and an understanding of the automation process.
- Any changes in the administration system will need to be added or changed in the automation software before the automation tests can be run.
- Different products may need to be catered for individually in the automation software, even when the same administration system is used for all products.
- It may not be possible or justifiable to automate certain functionalities of the administration system (for example, paper-based processes such as scanning in application forms or copies of ID documents).
- Automation testing cannot be conducted on an unstable system; it will only lead to extra costs being incurred and more effort required to redo the programming.