EDISON Software Development Centre
+7 (499) 500 14 94
http://www.edsd.com
market@edsd.com

Client-server application testing plan

1. INTRODUCTION

The present plan contains and describes testing strategy principles applied for remote access system testing. The plan is intended to be used by project executors and provides an understanding of project testing. This document assigns duties within the testing process and provides information on upcoming tests.

This testing plan has been developed to find solutions for the following problems:

- To plan test management and technical support within the whole system development life-cycle.
- To develop a complete testing plan, describing the nature and scope of the testing required to achieve the goals set and solve problems within the testing project.

1.1. Testing Aims

The main testing aims are:

- Fulfillment of all system requirements and criteria set for the software product.
- Increasing the probability that the application will function correctly in any situation and meet the set requirements by conducting thorough bug detection;
- Ensuring the functionality of each unit in accordance with unit specification requirements;
- Guaranteeing the functionality of the system as a whole in line with system specification requirements;
- System and unit fail-safe maintenance;
- Maintaining set efficiency parameters;
- The supply of high-quality raw materials and source codes;
- Promptly informing interested parties on regular assemblage quality levels;
- Providing the user with the most convenient graphical interface;

1.2. Testing strategies.

The main testing aims are:

- Test each unit and system component to ensure it functions in accordance with the operational requirements;
- Conduct system testing aimed at unit and component interaction in accordance with system requirements;
- Determine and maximally improve the performance of the system and every individual unit;
- Carry out stress testing in order to ensure the fail-safe operation of the system and each individual unit;
- Maximizing testing process automation;
- Development of a sufficient set of test suites for new unit and component testing;
- Well-timed development of test suites for error correction;
- Increasing test suite code coverage;
- User-convenience testing of units with graphical interfaces.

1.3. Testing types

In order to achieve the testing aims mentioned above, the following methods will be used. Testing itself is a multi-sided process and the testing types described below may overlap. A specific list of testing methods for every unit is provided for test work.

- *Analysis of specification requirements for each unit and component* – test setting preparation and identification for every single system component.
- *Analysis of software requirements specification* - test setting preparation and identification for the whole system.
- *Manual testing* – a tester manually conducts a testing cycle and registers all test results in a report.
- *Automated testing* – *an* automated testing cycle is conducted and interested parties are subsequently notified of the results automatically.
- *Complex testing* – manual and automated testing combined. The most popular testing method in practice.
- *Smoke testing* – the simplest form of testing based on the determination of system build success using the source code path that is under development. Normally conducted once a day.
- *Daily testing* - one of the steps of a testing cycle that must be conducted daily.

- *Unit testing* – **the most important form of testing**, based on checking functionality, methods and properties in conditions of correct and incorrect performance. This testing is conducted at the source code level of every existing class. Things to be tested during this stage:
  - correct class identification;
  - class structure correspondence to requirements specification;
  - Sufficient functionality of the class;
  - class is compatible with automated code processing tools (code documentation composition, coverage and code quality analysis, etc.);
  - incorrect functionality and errors are fixed;
  - class is compatible with interconnected classes within: use inheritance, polymorphism, call procedures, etc.;
  - Runtime, run frequency and resource load meet requirements;
  - class doesn't contain memory or other resource leaks;

Theoretically speaking, it is necessary to test every single line of source code. When the product is under initial development, fixing bugs is inexpensive. However, later the process of fixing such problems becomes more and more expensive. In most cases providing unit tests (which are usually created during the period of class development) is the developer's responsibility.

- *Integration testing* – after developing tests on separate classes, it's necessary to check their compatibility within one execution process. It's necessary to check the compatibility of classes that were designed by different developers using different tools. This form of testing is based on a previous one and is also developed at the source code level. Usually, test examples are based on component call.
- *End-to-end testing* – checks system component efficiency at the level of several single executive process interaction points. At this stage the functioning of the client-server system and its internal interaction with external components is tested. End-to-end testing is usually connected with processes that make test inquiries. This method helps to check macro level functions, reliability, performance, and coordination.
- *Functional testing* – considers products that consist of multiple classes, processes, components and data as single entities. At this stage the product's working capacity, functionality, technical characteristics, and business logic are checked. Such checks may be conducted using several equipment environment configurations and information sets.
- *Interface testing* – client and administrative UI check for performance in user scenarios. A user scenario is an operational sequence which imitates user activity while working with the system interface. Such scenarios must cover the requirements specification for the given UI. This type of testing is conducted in automatic mode with the help of special tools. Testing is aimed at checking UI performance with all possible screen settings (different screen resolution, zoom, font) and changes of focus while using a mouse and keyboard.
- *Stress testing* – identification and checking of system performance characteristics with a particular hardware configuration and information set.
- *Database testing* – checking of external database functionality and storage procedures in accordance with requirements specification. Checking the database access security policy in line with system functions. Identification and checking of database characteristics (performance, average access time, maximum number of clients, minimal and maximum query-processing time, etc).
- *Security testing* – determining functions and checking the list of system features available for each one. May be conducted at the interface, component, database, unit and network level. May only be conducted in accordance with the document "Security Policy". Testing includes a data encryption method check in terms of storage and transfer, access denial to prohibited functions, line-tapping, ID forgery, denial of service and other attacks.
- *UI usability testing* – usability report, assimilation speed, UI system visualization compilation. This report may contain information about the enhancement of these characteristics.
- *Hardware configuration testing* – system efficiency check with a particular hardware configuration and information set.
- *Verification* – checking a developer's success in fixing a bug. This is conducted by a tester in a testing environment.
- *Regression testing* – recurrent random product testing with modified sectors after fixing a bug or adding a new function. Source code changes may result in new bugs occurring within interdependent functions. This testing type

minimizes this kind of risk.

- *Inspections and critical reviews* – (un)scheduled testing of a system and its separate components in order to:
  - identify weak spots;
  - determine the degree of correspondence of standards and requirements;
  - identify development trends;
  - develop new architecture solutions;
  - come up with code refactoring suggestions;
  - improve qualitative and functional characteristics;
- *Source code analysis* – scheduled source code testing in order to identify the level of correspondence to the "Source code format requirements". Development of refactoring suggestions.
- *Code testing coverage analysis* – automatic detection of code areas which weren't involved in benchmark tests in order to develop new testing for coverage optimization.
- *Installation testing* – a performance check for installation packages, installation scripts for copying, updating and further automatic settings;
- *Documentation testing* – documentation checking for manual description comprehension in accordance with "Engineering documentation for users and administrators package development requirements ".
- *Final release testing* – testing that is conducted during the final stages before a product is released. Consists of two parts:
  - *Alpha*-testing - conducted in accordance with official requirements at the test site of the developer.
  - *Beta*-testing – final testing stage, which corresponds to everyday "real life" working conditions at the client's company.

1.4. Documentation

During the process of developing and checking test samples, it is necessary to create documentation. Such documentation can be accessed at any time and serves the purpose of informing the reader on the following issues:

- full list of tests;
- testing task for each component;
- list of tests for each component;
- every example must be clearly commented on;
- chronological archive of test results.

2. TESTING CYCLE

In this section a testing process is described that consists of the following types of activity in order of importance: urgent unscheduled activity, release testing, everyday scheduled activity, new test development, and semi-annual activity.

2.1. Urgent activity

Urgent activity deals with urgent instructions from a project manager to a tester. The majority of urgent instructions must be accomplished because if they are not the development process may be halted.

This type of activity also includes critical error verification and patching. A tester has to be very attentive during the verification process because any change affects other code areas and may result in the occurrence of new errors and bugs. It is necessary to carefully check the parts of the code that are assumed to be connected in order to bring the process closer to regression testing for each incident.

Managers have to understand that a great deal of urgent activity may result in a scheduled activity crash and this will, in turn, lead to less testing. Therefore, it is necessary to schedule the load and increase the pace of testing work.

2.2. Release testing

This process is temporary. It begins at the moment when final testing starts and ends right after the release has been accepted. Normally, the purpose of release testing is to bring out a given product with certain features at a particular time. A necessary attributes of this type of testing is the checking of a product's functional characteristics in accordance with requirements specification. In order to do this, specialists carry out a thorough system requirement specification analysis for every separate component and the system as a whole.

The most important thing is to compare the current release with the previous one and use the differences between the two

for regression testing.
A tester develops an installation package.
While release testing is being carried out, successful automatic daily testing, that has alpha and beta testing purposes, is also under way. Installation and documentation are tested and additional manual testing is conducted. In addition, clients may be consulted regarding questions concerning system transfer and deployment.
During release testing a tester may also discover new urgent incidents.

2.3. Daily activity

Daily activity testing consists of scheduled automatic testing on a test bench.
Every night an attempt to implement a system build using a source code is undertaken. Thus, "smoke" testing is conducted.
If the test is satisfactory, an automatic scheduled testing implementation with all possible configurations is carried out.
As a result, a report is compiled and sent to all interested parties. A smoke testing crash is regarded as a critical error and must be analysed.
This type of testing also includes non-critical error verification and patching. Every problem should be tested individually.
The problems that occur again should trigger additional testing.
A tester monitors the testing process and compiles a daily testing report. He/she manages the test launch, configures the test bench and testing environment.
A tester constantly analyses system requirements specification, technical tasks and regularly gives recommendations on enhancing the testing process. A tester also informs developers about discrepancies in functionality - both those that are described in the task, and those that relate to real system characteristics.
A tester also carries out additional manual testing that must be conducted. Actions that are repeated more than 3 times require further automation.
The task of daily activity testing is a form of constantly automated testing.
During the process of system enhancement, some test suites may cease to function correctly, for instance, during UI customization. Therefore, the task of everyday testing is to analyse the causes of failures, update and fix the test suites.
During the testing process, a tester discovers new incidents using test results and informs developers about them.

2.4. Developing new tests

New testing blocks are designed by a developer during system creation and patching. When a testing block is completed, a tester puts the results into an automatic testing task planner.
If so desired, a tester may also take part in the development process itself. A tester may take part in the initial development process of basic unit test selection, and may also compile a testing list during the process of test analysis and enhancement.
Normally testing responsibilities are distributed in the following way:

Automatic testing planner	Tester, developer
Smoke testing	Tester
Unit testing	Developer
Integration testing	Developer
End-to-end testing	Developer
Stress testing	Developer, tester
Configuration testing	Tester
Database testing	Developer, tester
Security testing	Developer, tester
Interface testing	Tester, Developer

2.5. Semi-annual activity

Semi-annual activity is conducted twice a year upon the request of a project manager or after a release date. As a result, a status report containing enhancement suggestions is compiled. This activity includes the following:

Inspection and critical code examination	General manager, developer, chief programmer
UI usability testing	Tester
Source code quality analysis	Tester

Testing code coverage analysis	Tester

3. TEST BENCH

3.1. Automation testing task planner

It is necessary to develop software using any scripting language that allows the tester to construct and go through the testing sequences in the time allotted.

This type of software performs daily integration, smoke testing and automatically launches setup testing blocks. Here, it is necessary to provide management support using several computers and various virtual devices.

The planner provides a UI for the compilation and presentation of reports on collected test results. Report data is comprised of a separate list of test suite results. If testing fails, the text is also saved and added to the report. The result is either a success or a code error.

A testing block is a minimal task. A testing block is carried out as a separate executive file which may contain unit, integration or stress tests.

It is necessary to develop a template for such files. The template should provide result testing scripts for the planner. It is necessary to develop an integration method for automatic GUI testing of user application interfaces.

After all testing has been completed, an automatic expert assessment is conducted. The report is analysed and final conclusions are drawn. In addition to this, a planner also provides the option of carrying out expert analysis. The total success rate (or percentage) of the entire testing process is then calculated.

The planner monitors launches, creates a launch log and saves the results of every attempt in a separate folder.

A finished report is sent to all interested parties by e-mail. This list can be configured.

3.2. Hardware configuration

For the system to function correctly the following testing server and client PC configurations are needed. Carrying out testing requires: a test bench that consists of two servers of configuration #1, one server of configuration #2, and two personal computers of configuration #2 and #3.

Configuration #1 (Server):

Configuration #2 (Client):

Configuration #3 (Remote client – minimal possible):

3.3. Configuration and data arrangement

The system is tested using a single environment – the one that the client uses:

- Server SRV1 with installed database. IP = x. There are 30 documents in a test database. The server configuration is: x.
- Server SRV1 with installed application server. IP = x. Server is installed in the folder x. The configuration is: x.
- Client PC CLI3 with installed client and administrative interfaces in folders x.
- Server SRV1 is directly connected to the server SRV2 via network interface 1 Gbps.
- PC CLI3 is connected to server SRV2 via a 100 Mbps switch and another 100 Mbps network interface.
- On server SRV2 and client PC CLI3 client and administrative interfaces may be launched.
- On server SRV2 a SQL Server database is installed in folder x, name instance x. Settings ODBC: x.
- Security firewalls are switched off.

3.4. Testing components

On the basis of this plan, testing tasks aimed at certain unit testing are set according to the framework of the Testing Plan Appendix. Here is a list of general components:

1.
2.
3.
4.

5.
6.
7.
8.
9.
10.
11.
12.
13.
14.

All testing procedures are conducted on a test bench, system components are tested using the following configurations:

1		№1, №2
2		№1, №2
3		№1, №2
4		№1, №2
5		№1, №2, №3
6		№1, №2, №3
7		№1, №2
8		№1, №2
9		№1, №2
10		№1
11		№1, №2, №3
12		№1, №2, №3
13		№1, №2, №3
15		№1, №2, №3

4. APPENDIX # 1

Testing tools:

1. Windows-embedded profiling and history facilities
2. Unit testing classes, testing block code
3. Automatic UI testing tool
4. Automation testing task planning
5. DevPartner
6. Microsoft Office
7. Windows Sysinternals Tools
8. Microsoft Debugger

List of necessary licenses:

№	Name	Quantity
1		
2		
3		
4		
5		

6		
7		
8		
9		
10		
11		
12		