

# Test-bed for Verification and Validation Activities in Developing an Operations Support System

Dae-Woo Kim, Hyun-Min Lim, and Sang-Kon Lee

Quality Improvement Research Division  
Network Technology Lab., Korea Telecom  
463-1, Jeonmin-Dong, Yusung-gu, Daejeon, 305-811, Korea

**Abstract** - *This paper describes a test-bed for verification and validation activities in KT-NeOSS (Korea Telecom New Operations Support System) development. In this paper, we show the test phases for performing verification and validation activities during the development and maintenance of KT-NeOSS. We describe the experiences and considerations in building the test-bed for the activities. With this test-bed, we performed various testing activities and succeeded in developing the KT-NeOSS.*

**Keywords:** OSS, Evaluation, Test-bed, Testing, Software Development Lifecycle

## 1 Introduction

In general, the models used for the software development lifecycle have been sequential, with the development progressing through a number of well defined phases. The sequential phases are usually represented by a V or waterfall diagram. These respective models are the V lifecycle model and waterfall lifecycle model [1]. In these models, we are particularly concerned with the testing and evaluation phases to improve the quality and reliability of the software system.

In KT-NeOSS, the individual operation support systems for facilities or services are integrated into a single system with a common platform. KT-NeOSS combines customers, workplaces, and management. This is the KT next-generation OSS, which opens the way for e-Network Operations. We call this the NeOSS (New Operations Support System). Since the KT-NeOSS was a big project, there were many risks involved during the development process. Thus, it was not appropriate to apply the general software development lifecycle to the KT-NeOSS development project. To ensure the success of this project, we exerted efforts to identify all defects and eliminate them as much as possible during the development process.

Therefore, in rigorously performing the testing and evaluation activities, we looked at the testing and evaluation phases based on the general software development lifecycle and referred to the quality model from ISO/IEC 9126 [2]. In this paper, the tests for NeOSS included code review, unit testing, integration testing and acceptance testing according to the general software development lifecycle. In addition, we performed a fail-over test and efficiency tests. Also, before we applied the developed NeOSS in the field, we conducted an operation test to improve reliability [6,7]. Also, in order to perform the verification and validation activities easily, we needed a test-bed that could provide the software development environment and the test environment for the developers and the testers. The test-bed should be able to provide the test environment for them for the maintenance of the operated software system after its field release. To meet these requirements for the test-bed, we looked at the considerations for the test-bed model before building it. By conducting the testing and evaluation activities with the test-bed, we were able to successfully develop the NeOSS and support its maintenance.

The rest of this paper is organized as follows: Section 2 gives a brief overview of the NeOSS. Section 3 looks at the general software development lifecycle related to the NeOSS. Section 4 describes the testing and evaluation activities for the NeOSS. Section 5 describes the test-bed models. Section 6 shows the built test-bed. Finally, we conclude our work and outline some future work in Section 7.

## 2 Overview of the NeOSS

The NeOSS is KT's integrated operations support system. Its main functions are service configuration, service assurance, service quality and operation information management, facility management, network management (IP (Internet Protocol), ATM (Asynchronous Transfer Mode) and etc.), and others. The services related to these are PSTN (Public

Switched Telephone Network), ADSL (Asymmetric Digital Subscriber Line), VDSL (Very-high-bit-rate DSL), Wireless Internet service, and others.

The core technologies of the NeOSS are the application architecture and the business process based on CBD (Component Based Development), multi-layered application architecture, common development platform (MS (Microsoft).NET), EAI (Enterprise Application Integration) for workflow and message bus, common data code, single sign-on, and others.

Figure 1 shows the architecture of NeOSS that connects the sub-systems through the EAI information bus. The sub-systems are SO (Service Ordering), SA (Service Assurance), FM (Facility Management), ADM (Access Domain Management), WM (Workforce Management), SLA (Service Level Agreement), and NM (Network Management). Also, according to the NeOSS development plan, the network quality information system and others are developed and new services added to the NeOSS.

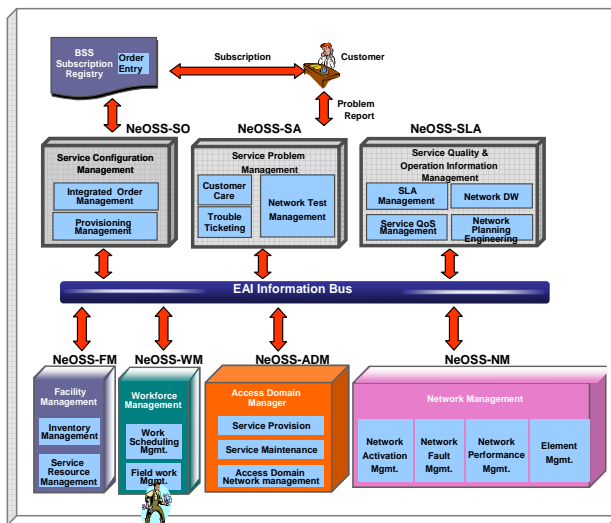


Figure 1. Architecture of the NeOSS

### 3 Software Development Lifecycle related to the NeOSS

This section describes the general software development lifecycle and shows how it was applied to the NeOSS during its development.

#### 3.1 V Lifecycle Model

The models used for the software development lifecycle were sequential, such as the V lifecycle model and the waterfall lifecycle model. Figure 2 shows the V lifecycle model [1]. The waterfall model is similar to

the V model. In fact, there are many variations of the V and waterfall lifecycle models, introducing different phases to the lifecycle and creating different boundaries between phases.

Among the phases in Figure 2, test activities are included from the Code and Unit Test to the Acceptance Test.

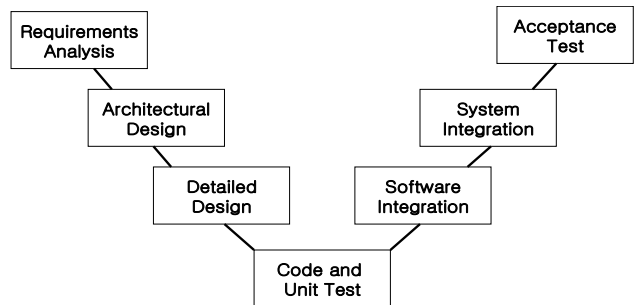


Figure 2. V Lifecycle Model

#### 3.2 Progressive Development Lifecycle Model

A common problem with software development is that the software is needed quickly, but takes a long time to fully develop. The solution is to form a compromise between timescales and functionality, providing for the “interim” deliveries of the software, with reduced functionality, but serving as stepping stones towards the fully functional software. The corresponding lifecycle model is referred to as the progressive development lifecycle and is shown in Figure 3 [1].

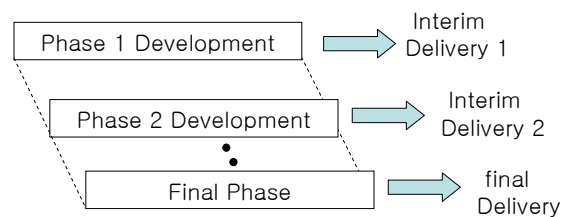
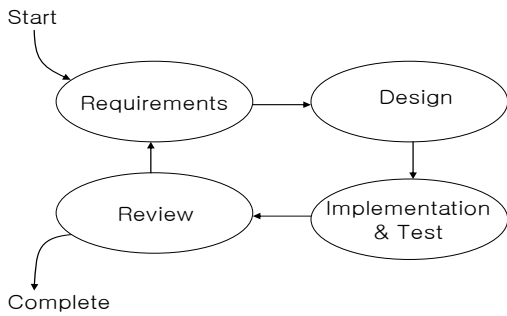


Figure 3. Progressive Development Lifecycle Model

#### 3.3 Iteration Lifecycle Model

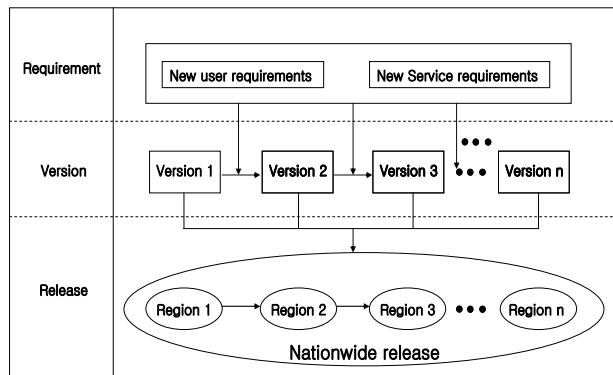
The iterative lifecycle model shown in Figure 4 does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just a part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model [1].



**Figure 4.** Iterative Lifecycle Model

### 3.4 Development Process for the NeOSS

The three abovementioned models were applied to the NeOSS development and its maintenance. During the development process, the NeOSS followed the V lifecycle model. During the field release, the NeOSS followed the progressive development lifecycle model because when the system was released from a regional area into the national area, new requirements and new services were added and developed. Also, during the verification and validation activities, the NeOSS followed the iterative lifecycle model in cases where the user or tester required new functions and modification of the requirements. This means that the three models were applied to the software system development in combination.



**Figure 5.** NeOSS Development Plan

Figure 5 shows the NeOSS development plan. The new user requirements and new service requirements from the business department were added continuously. The version was changed according to the continuously implementation of new requirements. The release was performed by regional area and extended to the national area.

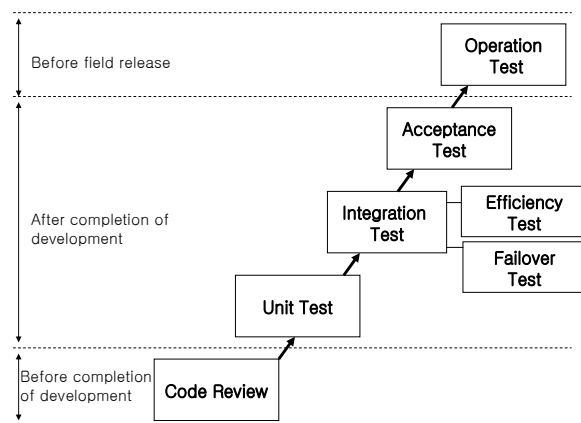
## 4 Testing and Evaluation Activities for the NeOSS

In order to manage the quality of the NeOSS, we conducted the testing and evaluation according to the phases based on the lifecycle models described in Section 3, which included the verification and validation activities in each of the phases. Before we dealt with the test phases in the development of the NeOSS, we took the following into consideration:

- The period of development is not enough to complete the NeOSS development.
- After the first version of the NeOSS is developed, we need to apply it to a regional area and then extend it to the national area.
- Our development framework is the MS.NET framework.

We focus on functionality, reliability, usability and efficiency among the external metrics of ISO/IEC 9126 [2,3] and reinforce the verification and validation processes [4] to successfully develop the NeOSS.

As shown in Figure 6, the phases are the testing and evaluation phases for the NeOSS. These testing and evaluation phases were applied to the NeOSS from the development to the field release and maintenance. The characteristics are shown in Table 1. Each of the phases focuses on the functional testing of the NeOSS. The tests for the phases referred to the black box tests, except for the Code Review. Also, the functionality test included the sub-characteristics from ISO/IEC 9126 such as suitability, accuracy, interoperability, security, and others.



**Figure 6.** Testing and Evaluation Phases for the NeOSS

**Table 1. Testing and Evaluation Phases**

Phases	Description
Code review	Testing for compliance with the development standard and the grammar of the program and module
Unit test	Testing for the respective sub-systems
Integration test	Testing for the integrated sub-systems by service
Acceptance test	Testing by the operation group and the users for their acceptance of the NeOSS
Operation test	Testing for NeOSS operation by the users before applying it to the field (including monitoring the performance)

The code review in Figure 6 was conducted when the NeOSS was being developed. In particular, the operation testing was performed to validate the suitability, performance, and stability by the users using the new system in practice. The purposes of the operation testing were as follows:

- Checking the functionality of the NeOSS by using the system functions such as handling the orders by service, the failures report, and others
- Performance monitoring by checking the performance metrics such as the CPU utilization for the servers, memory usage, number of Web requests per second, and others during a specified period under the functionality testing
- Analysis for the support activities of the NeOSS technical support team and Help desk for the NeOSS (This is in preparation for supporting the operation of the NeOSS.)
- Getting hold of users' opinions on the NeOSS

The following describes the tests related to the test phases.

**Functional Testing** This testing verified the functionality of the NeOSS according to the testing and evaluation phases, from the Unit test to the Operation test, except for the Code Review.

**Fail-over Testing** This testing was performed in the integration testing environment and verified the fail-over functions of server failures, which could take place under real operation. For example, the objects are as follows.

- L4 switch's load balancing function test when one of the web servers or one of the application servers is down
- Clustering function tests for the EAI-AP (Application) servers, EAI-DB (DataBase) servers, and DB servers

**Efficiency Testing** The kinds of this test are shown in Table 2

**Table 2. Efficiency Testing**

Tests	Description
Performance test	Checking the response time and the processing time
Load test	Checking if the system is stable when the maximum number of users are using it
Stress & Stability test	Checking if the system is stable under the specified overloading during a specified period of time

The tests on Table 2 by sub-systems were performed by the load-generating tool in the integration testing environment. To check the stability of the system, we measured the response time, CPU utilization, memory usage, amount of memory leaks, and others.

**Maintenance Testing** The maintenance tests for the NeOSS are as follows:

- Testing for bug-fixed and additional functions  
This tests the bug-fixed functions that were found during operation and the additional functions required by users.
- Testing for MS product patches  
This checks the stability of MS product patches such as MS.NET, MS SQL and MS BizTalk before they are applied to the NeOSS.
- Regression Testing  
This is to check if the patches and additional functions affect the existing functions. We used a tool that performs the test automatically in order to save on manpower and execution time of testing. In particular, we focused on the query functions of the core functions in the sub-systems. The reason is that using a test automation tool might need more manpower to make the scripts and to maintain them compared to manual testing [5].

During the test phases, we repeated the test phases to improve the functionality, reliability, usability, and efficiency of the NeOSS whenever the results showed failures, defects, and additional requirements from the testers and users.

## 5 Test-bed for the NeOSS

For the development and maintenance of the NeOSS, we needed a test-bed to perform the verification and validation activities according to the software development model in Section 3 and the testing and evaluation phases for the NeOSS in Section 4. In practice, the real operating system of the software system is the best test-bed, but the real operating system could not be used as a test-bed after the first release. Therefore, a test-bed should be built to manage the quality of the software system. Especially, in the case of the NeOSS, if the system has incorrect functions or system crashes or system performance degradation, the problems would create big problems for Korea Telecom's business operation. To prevent these problems, we needed to find and solve the defects and the hidden problems of the software system through the test phases before the field release. We looked at the considerable points for the test-bed. In the case of the NeOSS, the points are as follows:

- The development environment and the test environment should be separated from each other to evaluate the software accurately.
- The software version should be controlled by the version manager. When the software is evaluated, the version should not be changed by the developers.
- The development environment should be used by the developers only for the development and should not be used as a test environment by the testers to prevent the developers from influencing the testers' test.
- The test environment should provide the unit test, integration test, efficiency test and fail-over test environments for the testers and the users.
- The test environment should support the development and maintenance of the NeOSS.

We took consideration of the abovementioned points and the architecture of the test-bed environment is shown in Figure 7.

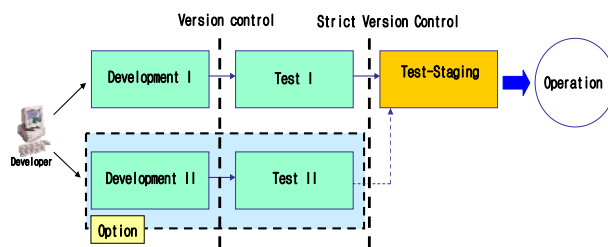


Figure 7. Architecture of the test-bed

The description of the environments is shown on Table 3

Table 3. The environments in the Test-bed

Environment	Description
Development I	<ul style="list-style-type: none"> <li>- Development environment.</li> <li>- Providing the code review and the unit test environment by the sub-system of the NeOSS for the developer.</li> <li>- This is built with essential equipment that provide the development environment for the developer.</li> <li>- The version control depends on the developers.</li> </ul>
Test I	<ul style="list-style-type: none"> <li>- Test environment.</li> <li>- Providing the unit test and the integration test environments of the NeOSS for the developers.</li> <li>- The version of the NeOSS is controlled by the version manager.</li> </ul>
Test-Staging	<ul style="list-style-type: none"> <li>- Test environment is similar to the operation environment.</li> <li>- Providing the unit test, the integration test, the efficiency test and fail-over test environments of the NeOSS for the tester and the user.</li> <li>- The version of NeOSS is strictly controlled by the version manager.</li> <li>- The same version as in the operation environment is kept in this environment.</li> </ul>
Development II (Optional)	<ul style="list-style-type: none"> <li>- Development environment for the next version, which has a big difference from the current operation version.</li> <li>- The environment's configuration and usage are the same as in Development I.</li> </ul>
Test II (Optional)	<ul style="list-style-type: none"> <li>- Test environment for the next version, which has a big difference from the current operation version.</li> <li>- The environment's configuration and usage are the same as in Test I.</li> </ul>

**Development I Environment** The Development I environment is usually mixed with the maintenance version and the next NeOSS version. Before we built the test-bed environment, we checked the developer's development environment and the version control method. The result is that they were using their PCs (Personal Computers) as the development environment to implement the functions and control their version by themselves. This is the reason why the MS.NET framework was used for the development of the NeOSS. Therefore, we built the development environment with essential equipment enable the developers to develop the interactive function between the sub-systems of the NeOSS and perform the unit testing in this environment.

**Test I Environment** Test I environment was independent from the Development I environment. In this environment, the version of the software system was controlled and managed by a version controller,

who used Visual Source Safe from Microsoft. Under the controlled version, the developers of the sub-systems can perform the unit testing and integration testing.

**Test-Staging Environment** This test environment was the same version as that in the operation environment. In this environment, the new versions, which were strictly controlled, were tested by the tester and users before they are finally applied to the operation environment. Therefore, we called this environment Test-Staging. This environment's hardware configuration was similar to that of the operation environment but the scale was smaller, and the same vendor's hardware as that in the operation environment was used to prevent problems from the variation in hardware characteristics among different vendors.

**Development II and Test II Environments** These environments are for the next version of the NeOSS, which has a big difference from the operational version of the NeOSS. But these environments are optional and only made when they are needed. In addition, the building cost for the two environments is expensive. Also, the Development II environment is built with essential equipment like in Development I because new services are continuously required by the business department. However, the building cost of the Test II environment is expensive.

To solve the cost problems in building the environment, Test II was built by temporarily changing the Test-Staging environments to Test II. The example is shown in Figure 8.

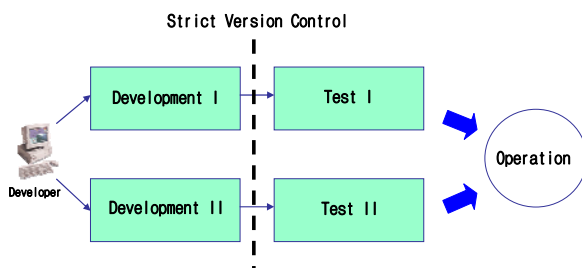


Figure 8. Temporary test-bed environment

After the new version and the current version were integrated, the test-bed environments were reallocated, as shown in Figure 9.

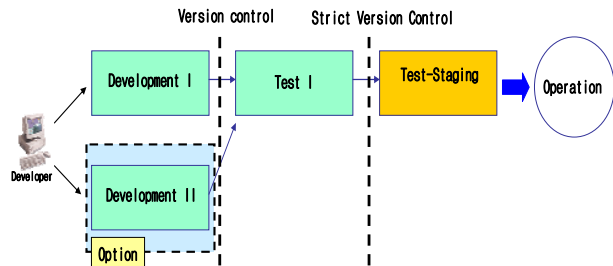


Figure 9. The reallocated test-bed environment

As in Figure 8 and Figure 9, we temporarily used the Test-Staging as the Test II.

## 6 Building the Test-Bed for the NeOSS

To build the test-bed for the NeOSS, we used the essential equipment for the environments. The servers allocated by the test-bed environment are shown on Table 4.

Table 4. Servers for the test-bed environments

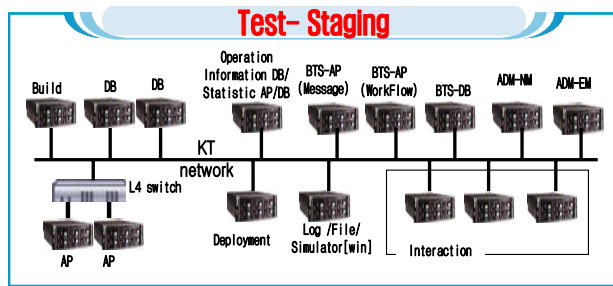
Environment	Servers
Development I Development II	AP (Application) server, DB (Database) server, EAI-AP server, EAI-DB server, Interaction server,
Test I	AP server, DB server, EAI-AP server, EAI-DB server, Interaction server, ADM-NM (Access Domain Management – Network Management) server, ADM-EM (Element Management) server, File server, Log server, Simulator server, Build server, Deployment server, Statistics AP server, Statistics DB server, Control & Monitoring server. MOM (Microsoft Operations Manager) server.
Test-Staging	AP server, DB server, EAI-AP server, EAI-DB server, Interaction server, ADM-NM (Access Domain Management – Network Management) server, ADM-EM (Element Management) server, File server, Log server, Simulator server, Build server, Deployment server, Statistics AP server, Statistics DB server

On Table 4, if one server could perform various functions, we used one server. For example, if we needed two servers as a log server and a file server, we used one server for both of them. The Control & Monitoring server and the MOM server were included just as in the Test I environment because they are used for the development and testing of the functions concerned with the NeOSS 's operation. Therefore, they were not included in Test-Staging environment.

We made the Test-Staging environment to be similar to the operational environment as much as possible, including the hardware configuration of the

operational environment and the kind of servers. For example, 64-bit servers were using different operating systems (Windows Server 2003, Datacenter Edition) from hardware vendors. To minimize the problem that might occur from using different hardware as those in the operational environment, we built the Test-Staging with the server from the same vendor with the operation environment.

The DB storage size of the Development I and Test I environments was the maximum required for performing the functional testing. The DB storage size of Test-Staging was the same size as that in the operational environment to partially perform the performance testing of the NeOSS. Figure 10 shows the Test-Staging environment.



**Figure 10.** Test-Staging environment

In Figure 10, the servers are 32-bit machines, except for the 64-bit DB servers. The DB server's scale for Test-Staging is about 9% of the operational environment and the AP server's scale for Test-Staging is about 6% of the operational environment.

## 7 Conclusions

In this paper, we showed the verification and validation activities for empirically developing the KT-OSS, which was a big software development project, and our experiences in building the test-bed. To manage the quality of the software system during the development process and maintenance process, we needed a test-bed. And to build the test-bed, we first looked at the test phases of the NeOSS, the development environment, the test environment, and the version management in the NeOSS development process. We built a test-bed that could provide the environments to be able to successfully conduct the development and testing of the NeOSS by the developers and the testers. The benefits of the test-bed are as follows:

- Providing the development environment and the maintenance environment for the developers and the testers
- Strict control of the version
- Improvement of the NeOSS reliability by managing and controlling the quality of the system

In particular, by keeping the Test-Staging with the same version as that in the operational environment, we were able to find the defects and hidden problems of the developed system. This improved the reliability of the system before the developed version was released. Also, by separating the development environment and the test environment, we could strictly control the version. Through the use of this test-bed, we successfully developed and released the NeOSS. At present, the NeOSS is used by about 30,000 users

In the near future, we need to study methods for performing the efficiency testing in the test-staging as much as predicting the actual performance of the system in the operational environment because the test-staging environment is focused on functional testing.

## References

- [1] IPL Information Processing Ltd, "Software Testing and Software Development Lifecycles", 1996
- [2] ISO/IEC TR 9126-2: Software engineering-Product quality-Part2: External metrics, 19-12-2000
- [3] Shahid Nazir Bhatti, "Why Quality? ISO 9126 Software Quality Metrics (Functionality) Support by UML Suite", ACM SIGSOFT Software Engineering Notes, March 2005
- [4] Raghu Singh, "INTERNATIONAL STANDARD ISO/IEC 12207 SOFTWARE LIFE CYCLE PROCESS", Federal Aviation Administration
- [5] Brian Marick Testing Foundations, "Classic Testing Mistakes", 1997
- [6] Linda Rosenberg, Ted Hammer, Jack Shaw, "Software Metrics and Reliability", November 1998
- [7] [http://www.ece.cmu.edu/~koopman/des\\_s99/sw\\_reliability/](http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/)