Web-SynDic

Web System for Demonstrating the Syntactic Algorithms for Solving Linear Equations in Nonnegative Integers (Nonnegative Linear Diophantine Equations)

Test Plan

Department of Computer Science, Petrozavodsk State University, Russia

15th November 2004

Contents

1 General Description

This document is a Test Plan for the Web-SynDic Project. It describes the testing approach and test cases to be used for verification and validation of the web system. The testing is mainly focused on features and functions, which are listed in the User Requirements.

Related Documents:

1. Requirements Specification Document
2. Design Document

2 Approach

2.1 Features to be Tested

The following is a list of high-level functions that will be tested.

- Generating ANLDE (test ANLDE systems or ANLDE systems sets).
- Processing (solving) ANLDE.
- Producing reports on solution.
- User session processing.
- User registration and login features.
- Management of user data.
- Writing notes.
- System administrator controls:
  1. management of the web system;
  2. management of users;
  3. activity statistics of the web system.

This list directly based on the section Use cases of the Requirements Specification.

2.2 Environment Requirements

The minimal configuration, with which the web system is guaranteed to work, is defined in the Configuration Requirements. The team will use this one as a test environment.

2.3 Overall Strategy

Three phases for testing are used: unit, intergation and validation testing. For each of them, series of test cases are designed to exersise systematically the web system: its internal logic, required function, result behaviour and available performance.
Unit tests exersice that each individual unit works as it is supposed. These tests are executed by the developers in parallel with the implementation.
Integration tests exersice systematically all key combinations of subsystems to uncover errors and defects associated with subsystem interfaces. Bottom-up incremental integration is used.
Validation tests exersice on the behavior and the advanced functionality of the whole web system. User scenarios will be executed against the web system. They can be divided into:

Functional tests (see Functions of the web system) will test the integrated system and verify that it meets the requirements and criteria defined in the requirements document.

Performance tests (see Attributes of the web system: Performance) will ensure that the web system performance meets the specified performance criteria. This kind of testing should be some kind of stress testing.

Security tests (see Attributes of the web system: Security) should test the security of the web system. Nevertheless in the Project, they will not be performed as a dedicated part of testing, due to the following reasons. 1) Security of the web system is partially solved with the design decisions (e.g. technology of think web client). 2) External components, used with the web system, such as servlet/JSP container and Java tools, are considered to support the required security.

Acceptance tests have been defined in the Requirements Specification; they must be executed in accordance with Section Validation Criteria.

2.4 Criteria

Suspension / Exit Criteria. If any defect is found that seriously impact the test progress, then it is a reason to suspend testing. Criteria that justify the test suspension are:

1. Source code contains one or more critical defects that seriously prevent or limit testing progress.

2. Assigned test resources are not available when needed by the test team.

Resumption Criteria. If testing is suspended, resumption will only occur when the problem that caused the suspension has been resolved. When a critical defect is the cause of the suspension, the fix must be verified before testing is resumed. Some kind of regression testing must also be executed.
Completion Criteria. All phases of testing are considered as completed when:

1. all test cases have been executed without any error or defect;

2. system function, behaviour and performance meet the requirements;

3. acceptance testing has been done; the results were analysed and approved by the Customer.

3 Unit Testing

The used white-box testing methods include some kind of a coverage, such as branch coverage or independent path coverage. The test must exercise local data structures, boundary conditions, standard and error handling paths.

This unit test specification comprises a minimally needed description of the unit and test cases to exercise the unit.

For each test case one must define:

1. What the test case actually exercise, in terms of the functionality of the unit.

2. The input for the unit, including values of any external data that are read by the unit.

3. The expected outcome.

All tests for a unit are divided into the following groups.

1. **General:** execution of the unit in the simplest way that possible.

2. **Positive tests:** the unit does anything that is supposed.

3. **Negative tests:** the unit does nothing that is not specified.

4. **Special considerations:** exercise issues like performance or behavior.

These groups should exercise the internal logic of the tested unit to cover all its key functions, branches of code, states of variables (range and boundary testing), and data flows.

Unit test case design and specification is the responsibility of the author of the unit.

Java class is chosen as a basic test unit, so “class” will be used instead of “unit” below.

3.1 Web server pages and servlets

Author: Michail A. Kryshen'.
Parent subsystem: Web server.
Description: JSP pages and servlets that implement the web server.
Test cases: The following types of test cases will be used for web server pages and servlets.
Layout: All elements are located properly on the page (as was designed).
Input data: All input data are read and transmitted properly.
Output data: All output data are transmitted and visualized properly.
HTML validation: HTML code is valid HTML 4.01. A computer aided validator should be used.

3.2 Class ANLDEParser

Author: Andrew Salo
Parent subsystem: Web server
Description: This class is for parsing ANLDE systems or their sets from user input format (close to traditional math style in txt mode) and then connverting the systems into internal format (ANLDERequest object).

3.2.1 parseANLDE(source, set, limits)

Translates a test ANLDE system or an ANLDE system set from the ANLDE System Format or ANLDE System Set Format to the internal format (instance of the ANLDERequest class). ANLDEFormatException is thrown in case of parsing error or when an ANLDE system does not conform to the user limits.
Parameters:

1. String source: ANLDE system or set of ANLDE systems as a text string.
   Restrictions: The ANLDE System (Set) Format must be satisfied.
2. boolean set: flag to distingush the cases of one ANLDE system and a set of them.
   Restrictions: Possible values: true—source contains set of ANLDE systems, false—source contains single ANLDE system.
3. Limits limits: user limits.
   Restrictions: non-negative values in class Limits.

Return value: new instance of class ANLDERequest.
Dependencies: Class ANLDERequest, ANLDEFormatException, Limits
List of tests:

Test case 3.2.1.1 (general)**.** Translate a simple (2x4)-test ANLDE system.
Input data:
source:
```
x1 + x2 = 2*x1 + 3*x3
x3 + x4 = x1 + 2*x4 + x3
```
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object: n=2; m=4; names=(x1,x2,x3,x4); unknowns=(1,1,2,2); matrix=$\begin{matrix}2 & 0 & 3 & 0\\ 1 & 0 & 1 & 2\end{matrix}$

Test case 3.2.1.2 (general)**.** Translate a simple (3x6)-test ANLDE system.
Input data:
source:
```
x1 + x2 = 2*x6 + 3*x3
x3 + x4 = x1 + 2*x2 + x3 + 4*x5
x5 + x6 = 2*x6 + 3*x4
```
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object: n=3; m=6; names=(x1,x2,x6,x3,x4,x5); unknowns=(1,1,3,2,2,3); matrix=$\begin{matrix}0 & 0 & 2 & 3 & 0 & 0\\ 1 & 2 & 0 & 1 & 0 & 4\\ 0 & 0 & 2 & 0 & 3 & 0\end{matrix}$

Test case 3.2.1.3 (positive)**.** Translate a test ANLDE system with one equation.
Input data:
source:
```
x1 + x4 = x3 + 5*x2
```
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object: n=1; m=4; names=(x1,x4,x3,x2); unknowns=(1,1,1,1); matrix=$\begin{matrix}0 & 0 & 2 & 6\end{matrix}$

Test case 3.2.1.4 (positive)**.** Translate a test ANLDE system with one equation and empty right-hand side.
Input data:

source: `x + y + z = 0`
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=4, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object: n=1; m=3; names=(x,y,z); unknowns=(1,1,1); matrix=`0 0 0`

Test case 3.2.1.5 (negative)**.** Translate a test ANLDE system with double apperance of an unknown in a left-hand side.
Input data:
source:
```
x1 + x2 = 2*x1 + 3*x3
x3 + x1 + x4 = x1 + 2*x4 + x3
```
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=4, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
Exception on incorrect ANLDE system.

Test case 3.2.1.6 (negative)**.** Translate a test ANLDE system with double apperance of an unknown in a left-hand side in the same equation.
Input data:
source:
```
x1 + x2 = 2*x1 + 3*x3
x3 + x4 + x4 = x1 + 2*x4 + x3
```
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=4, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
Exception on incorrect ANLDE system.

Test case 3.2.1.7 (negative)**.** Translate a test ANLDE system with apperance of an unknown in a left-hand side with nonunit coefficient.
Input data:
source:
```
x1 + x2 = 2*x1 + 3*x3
x3 + 2*x4 + x4 = x1 + 2*x4 + x3
```
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=4, unknowns=6, systems=20, solutions=100, reportSolutions=100

generator: any
Expected outcome:
Exception on incorrect ANLDE system.

Test case 3.2.1.8 (specific)**.** Translate a large (equations, unknowns, coefficients) test ANLDE system.
Input data:
source: large ANLDE system in the correct format (see file `./ANLDEParser/LargeANLDESystems.txt`)
set: false
limits: time=100, memory=3000, coefficients=1000, solutionValues=10000, equations=4-50, unknowns=20-100, systems=20, solutions=1000, reportSolutions=1000
generator: any
Expected outcome:
correct ANLDERequest object.

Test case 3.2.1.9 (specific)**.** Translate a large (equations, unknowns, coefficients) NLDE system that is not ANLDE system.
Input data:
source: large ANLDE system in the incorrect format (see file `./ANLDEParser/LargeInvalidANLDESystems.txt`)
set: false
limits: time=100, memory=3000, coefficients=1000, solutionValues=10000, equations=4-50, unknowns=20-100, systems=20, solutions=1000, reportSolutions=1000
generator: any
Expected outcome:
Exception on incorrect ANLDE system.

Test case 3.2.1.10 (specific)**.** Translate a test ANLDE system which contains comments and blank lines.
Input data:

source:
```
# A test ANLDE system
  # some comment

######################
# A first equation!  #
######################
x1 + x2 = 2*x6 + 3*x3

x3 + x4 = x1 + 2*x2 + x3 + 4*x5         # end of line comment
x5 + x6 = 2*x6 + 3*x4
```
set: false
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object: n=3; m=6; names=(x1,x2,x6,x3,x4,x5);
unknowns=(1,1,3,2,2,3); matrix=
```
0 0 2 3 0 0
1 2 0 1 0 4
0 0 2 0 3 0
```

Test case 3.2.1.11 (general)**.** Translate a set of 3 simple test ANLDE systems.
Input data:
source: set of 3 simple test ANLDE systems
set: true
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=3, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object containing ANLDE system set.

Test case 3.2.1.12 (general)**.** Translate a set of one test ANLDE system.
Input data:
source: a test ANLDE system
set: true
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=3, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object containing ANLDE system set of one ANLDE system.

Test case 3.2.1.13 (positive)**.** Translate a typical (medium) set of medium test ANLDE systems.
Input data:
source: a set of test ANLDE systems
set: true
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object containing ANLDE system set.

Test case 3.2.1.14 (negative)**.** Translate a set of test ANLDE systems which contains non ANLDE systems (double apperance of an unknown in a left-hand side, double apperance of an unknown in a left-hand side in the same equation, apperance of an unknown in a left-hand side with nonunit coefficient).
Input data:
source: a set of test ANLDE systems which contains non ANLDE systems
set: true
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
Exception on incorrect ANLDE system set.

Test case 3.2.1.15 (negative)**.** Translate a set of test ANLDE systems with systems limit exceeded. Translate a set of test ANLDE systems which contains ANLDE system with (unknowns, equations, coefficients) limit exceeded.
Input data:
source: a set of test ANLDE systems with exceeded limits
set: true
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
Exception on incorrect ANLDE system set.

Test case 3.2.1.16 (specific)**.** Translate a large (systems) set of large (equations, unknowns, coefficients) test ANLDE systems.
Input data:

source: large ANLDE system set in the correct format
set: true
limits: time=100, memory=3000, coefficients=1000, solutionValues=10000, equations=50, unknowns=100, systems=20, solutions=1000, reportSolutions=1000
generator: any
Expected outcome:
correct ANLDERequest object containing ANLDE system set.

Test case 3.2.1.17 (specific)**.** Translate a large (systems) set of large (equations, unknowns, coefficients) ANLDE systems in the incorrect format.
Input data:
source: large ANLDE system set in the incorrect format
set: true
limits: time=100, memory=3000, coefficients=1000, solutionValues=10000, equations=50, unknowns=100, systems=20, solutions=1000, reportSolutions=1000
generator: any
Expected outcome:
Exception on incorrect ANLDE system set.

Test case 3.2.1.18 (specific)**.** Translate a test ANLDE system set which contains comments and blank lines.
Input data:
source: a set of test ANLDE systems
set: true
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct ANLDERequest object containing ANLDE system set.

Test case 3.2.1.19 (specific)**.** Translate a test ANLDE system set containing systems without equations.
Input data:
source: a set of test ANLDE systems containing systems without equations
set: true
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
Exception on incorrect ANLDE system set.

3.3 Class GeneratorSpooler

Author: Kirill A. Kulakov.
Parent subsystem: Algorithm server.
Description: Class for generating sequantially ANLDE systems.
Class Interfaces: None.

3.3.1 generateANLDESystem(anlde, limits, generator)

The task for generating a test ANLDE system is pushed into buffer and waits its turn. Class generator is used then for generation with the given limits and generator.
Parameters:

1. ANLDE anlde will contain the generated test ANLDE system.
   Restrictions: Output parameter. The system must satisfy the ANLDE definition. Generator does not generate a ANLDE system when the limits can not be satisfied by this generator or the limits are incorrect.

2. Limits limits contain user limits.
   Restrictions: Transite input parameter. Only nonnegative values are used.

3. String generator is a name of the selected generator.
   Restrictions: Transite input parameter Only names form the fixed set are used.

Return value: It does not return any value; but after generating the ANLDE system, the signal is sent and the system is stored in anlde object.
Dependencies:
Classes: Generator, GenTask, ANLDE, Limits.
List of tests:

Test case 3.3.1.1 (general)**.** Typical nonempty buffer.
Input data:
Buffer is nonempty (1-4 tasks).
Expected outcome:
The task was pushed into the buffer.

Test case 3.3.1.2 (positive)**.** Empty buffer.
Input data:
The Buffer is empty (0 tasks).
Expected outcome:
The task was pushed into the buffer.

Test case 3.3.1.3 (negative)**.** Buffer is full.
Input data:
The buffer contains 5 tasks.
Expected outcome:
The task must be correctly rejected.

Test case 3.3.1.4 (negative)**.** Buffer is corrupted.
Input data:
Buffer contains more tasks than it is allowed (tasks=6-10) or negative number of tasks.
Expected outcome:
Error recovery must occur. The task must be rejected.

3.3.2 generateANLDESystemSet(anlde, limits, generator)

The task for generating ANLDE systems set is pushed into buffer and waits its turn. Class generator is used then for generation with the given limits and generator.
Parameters:

1. ANLDE anlde will contain the generated set of test ANLDE systems.
   Restrictions: Output parameter. The system must satisfy the ANLDE definition. Generator does not generate a ANLDE system when the limits can not be satisfied by this generator or the limits are incorrect.

2. Limits limits. user limits.
   Restrictions: Transite input parameter. Only nonnegative values are used.

3. String generator. Name of the selected generator.
   Restrictions: Transite input parameter. Only names form the fixed set are used.

Return value: It does not return any value; but after generating the ANLDE systems set, the signal is sent and the set is stored in anlde object.
Dependencies:
Classes: Generator, GenTask, ANLDE, Limits.
List of tests:

3.3.3 getBufferCount(number)

The method returns the number of tasks (generation) in the buffer, i.e. the current state of this buffer.
Parameters: none.
Return value: int number is a count of tasks in the buffer.

Dependencies: Classes: GenTask
List of tests:

Test case 3.3.3.1 (general)**.** Typical nonempty buffer.
Input data:
Buffer is nonempty (1-5 tasks).
Expected outcome:
Correct number of tasks.

Test case 3.3.3.2 (positive)**.** Empty buffer.
Input data:
Buffer is empty (0 tasks).
Expected outcome:
Number of tasks=0.

Test case 3.3.3.3 (negative)**.** Buffer is corrupted.
Input data:
Buffer contains more tasks than it is allowed (tasks=6-10) or negative number of tasks.
Expected outcome:
Error recovery must occur.

3.4 Class SolverSpooler

Author: Kirill A. Kulakov
Parent subsystem: Algorithm server
Description: Class for solving sequantially ANLDE systems.
Class Interfaces: None.

3.4.1 solveANLDE(anlde, limits, solvers)

The task of solving a test ANLDE system or a set of ANLDE systems is pushed to the task buffer. Class Solver for solving with the given limits and solvers.
Parameters:

1. ANLDE anlde: a test ANLDE system or a set of them.
   Restrictions: correct ANLDE system or set.

2. Limits limits: user limits.
   Restrictions: nonnegative values in object Limits.

3. String[] solvers: used solvers (basic and alternative if any).
   Restrictions: use only known solvers.

Return value: The method does not return any value, but after solving it sends signal with solver outcome.
Dependencies: Classes: ANLDE, Limits, SolverOutcome, SolTask, Solver
List of tests:

Test case 3.4.1.1 (general)**.** Typical nonempty buffer.
Input data:
Buffer is nonempty (1-4 tasks).
Expected outcome:
The task was pushed into the buffer.

Test case 3.4.1.2 (positive)**.** Empty buffer.
Input data:
The Buffer is empty (0 tasks).
Expected outcome:
The task was pushed into the buffer.

Test case 3.4.1.3 (negative)**.** Buffer is full.
Input data:
The buffer contains 5 tasks.
Expected outcome:
The task must be correctly rejected.

Test case 3.4.1.4 (negative)**.** Buffer is corrupted.
Input data:
Buffer contains more tasks than it is allowed (tasks=6-10) or negative number of tasks.
Expected outcome:
Error recovery must occur. The task must be rejected.

3.4.2 getBufferCount(number)

The method returns the number of tasks (solving) in the buffer, i.e. the current state of this buffer.
Parameters: none.
Return value: int number is a count of tasks in the buffer.
Dependencies: Classes: GenTask
List of tests:

Test case 3.4.2.1 (general)**.** Typical nonempty buffer.
Input data:
Buffer is nonempty (1-5 tasks).

Expected outcome:
Correct number of tasks.

Test case 3.4.2.2 (positive)**.** Empty buffer.
Input data:
Buffer is empty (0 tasks).
Expected outcome:
Number of tasks=0.

Test case 3.4.2.3 (negative)**.** Buffer is corrupted.
Input data:
Buffer contains more tasks than it is allowed (tasks=6-10) or negative number of tasks.
Expected outcome:
Error recovery must occur.

3.5 Class Generator

Author: Kirill A. Kulakov
Parent subsystem: Algorithm server
Description: Class for sequential generating ANLDE systems.
Class Interfaces: None.

3.5.1 generate(anlde, limits)

The method generates a test ANLDE system calling appropriate generator. Generator's outcome is translated by class GeneratorOutputParser.
Parameters:

1. ANLDE anlde: to store ANLDE system and send signal

   Restrictions: none.

2. Limits limits: user limits

   Restrictions: non-negative values in class Limits.

Return value: Generated anlde object (or null on error).
Dependencies: Classes ANLDE, Limits, GeneratorOutputParser.
List of tests:

Test case 3.5.1.1 (general)**.** Generate a simple test ANLDE system.
Input data:
anlde: null

limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
generator: any
Expected outcome:
correct (3x6)-ANLDE system as anlde object, the signal was sent.

Test case 3.5.1.2 (positive). Generate a standard test ANLDE system.
Input data:
anlde: null
limits: time=100-300, memory=1000-10000, coefficients=5-1000, solutionValues=100-1000, equations=2-20, unknowns=3-20, systems=1-100, solutions=1-1000, reportSolutions=1-500
generator: any
Expected outcome:
correct ANLDE system as anlde object, the signal was sent.

Test case 3.5.1.3 (negative). Try to generate a test ANLDE system with incorrect limits.
Input data:
anlde: null
limits: time=0-10, memory=0-10000, coefficients=0-1000, solutionValues=0-1000, equations=0-20, unknowns=0-20 (equations>unknowns), systems=0-100, solutions=0-500, reportSolutions=0-500
generator: any
Expected outcome:
The ANLDE system has not been generated: 1) wrong user limits (reference to concrete error); 2) the generator cannot generate such a system.

Test case 3.5.1.4 (negative). Try to fail a generator.
Input data:
The generator must fail.
anlde: null
limits: time=10, memory=10000, coefficients=100, solutionValues=1000, equations=3, unknowns=6, systems=10, solutions=100, reportSolutions=100
generator: any
Expected outcome:
ANLDE system must not be generated due to a generator fail.

3.5.2 generateSet(anlde, limits)

The method generates a set of ANLDE systems calling appropriate generator. Generator's outcome is translated by class GeneratorOutputParser.

Parameters:

1. ANLDE anlde: to store ANLDE systems set and send signal

   Restrictions: none.

2. Limits limits: user limits

   Restrictions: non-negative values in class Limits.

Return value: Generated set ANLDE anlde or null on error.
Dependencies: Class ANLDE, Limits, GeneratorOutputParser.
List of tests:

Test case 3.5.2.1 (general). Generate a simple set of test ANLDE systems.
Input data:
anlde: null
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=5, solutions=100, reportSolutions=100
generator: any
Expected outcome:
The set with 5 correct test ANLDE system as anlde object, the signal was sent.

Test case 3.5.2.2 (positive). Generate a set with one ANLDE system.
Input data:
anlde: null
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=1, solutions=100, reportSolutions=100
generator: any
Expected outcome:
The set with the only correct test ANLDE system as anlde object, the signal was sent.

Test case 3.5.2.3 (negative). Fail in a generator.
Input data:
anlde: null
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=3, unknowns=6, systems=3, solutions=100, reportSolutions=100
generator: any
Expected outcome:
The set cannot be generated due to fail in generator.

Test case 3.5.2.4 (special)**.** Try to generate a large set.
Input data:
anlde: null
limits: time=100, memory=10000, coefficients=100, solutionValues=1000, equations=1-100, unknowns=1-100, systems=50-1000, solutions=100-1000, reportSolutions=100-500
generator: any
Expected outcome:
The set correct set of ANLDE systems or fail to clear reason.

3.6 Class Solver

Author: Kirill A. Kulakov
Parent subsystem: Algorithm server
Description: Class for sequantial solving ANLDE systems or their sets.
Class Interfaces: none.

3.6.1 solve(anlde, limits, solvers)

The method solves a test ANLDE system or a set of ANLDE systems calling appropriate solver. Solver's outcome is translated by class SolverOutputParser.
Parameters:

1. ANLDE anlde: a test ANLDE system or a set of them.
   Restrictions: correct ANLDE system or set.
2. Limits limits: user limits.
   Restrictions: nonnegative values in object Limits.
3. String[] solvers: used solvers (basic and alternative if any).
   Restrictions: use only known solvers.

Return value: SolverOutcome outcome.
Dependencies: Classes: ANLDE, Limits, SolverOutcome, SolverOutputParser.
List of tests for a test ANLDE system:

Test case 3.6.1.1 (general)**.** Solve a simple test ANLDE system.
Input data:
anlde: very simple ANLDE system
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=2-3, unknowns=4-6, systems=20, solutions=100, reportSolutions=100
solver: any

Expected outcome:
correct Hilbert basis or a particular solution was found. The signal was sent. The results of different solvers are similar.

Test case 3.6.1.2 (positive)**.** Solve a medium test ANLDE system.
Input data:
anlde: medium ANLDE system
limits: time=100, memory=10000, coefficients=10-50, solutionValues=1000, equations=3-10, unknowns=4-15, systems=20, solutions=500, reportSolutions=500
solver: any
Expected outcome:
correct Hilbert basis or a particular solution was found. The signal was sent. The results of different solvers are similar.

Test case 3.6.1.3 (negative)**.** Try to solve wrong ANLDE system.
Input data:
anlde: incorrect ANLDE system: non-ANLDE, no satisfaction with the limits, not one ANLDE system but a set of them
limits: time=100, memory=10000, coefficients=10-100, solutionValues=1000, equations=3-10, unknowns=4-15, systems=20, solutions=500, reportSolutions=500
solver: any
Expected outcome:
Solver must not solve such an ANLDE system.

Test case 3.6.1.4 (special)**.** Hard ANLDE system.
Input data:
anlde: large ANLDE system: many solutions, many equations/unknowns, large coefficients
limits: time=180, memory=15000, coefficients=10-1000, solutionValues=10000, equations=3-100, unknowns=6-150, systems=20, solutions=1500, reportSolutions=500
solver: any
Expected outcome:
ANLDE solver must solve these systems; other solvers may fail.

List of tests for ANLDE systems set:

Test case 3.6.1.5 (general)**.** Solve a set of simple test ANLDE systems.
Input data:
anlde: 3-set of very simple ANLDE system
limits: time=100, memory=10000, coefficients=10, solutionValues=100, equations=2-3, unknowns=4-6, systems=3, solutions=100, reportSolutions=100

solver: any
Expected outcome:
correct data for the report on solution The signal was sent. The results of different solvers are similar.

Test case 3.6.1.6 (general). Solve a set with the only test ANLDE system.
Input data:
anlde: medium ANLDE system
limits: time=100, memory=10000, coefficients=10-50, solutionValues=1000, equations=3, unknowns=6, systems=1, solutions=500, reportSolutions=500
solver: any
Expected outcome:
Correct solvers' outcomes. The signal was sent. The results of different solvers are similar.

Test case 3.6.1.7 (positive). Solve a typical (medium) set with medium test ANLDE system.
Input data:
anlde: medium ANLDE system
limits: time=100, memory=10000, coefficients=10-50, solutionValues=1000, equations=3-7, unknowns=6-14, systems=2-20, solutions=1000, reportSolutions=500
solver: any
Expected outcome:
Correct solvers' outcomes. The signal was sent. The results of different solvers are similar.

Test case 3.6.1.8 (negative). Try to solve a set with wrong ANDLE systems.
Input data:
anlde: 1) the set contains one incorrect ANLDE system (first, last, middle): non-ANLDE, no satisfaction with the limits, not one ANLDE system but a set of them; 2) the set contains several incorrect ANLDE system
limits: time=100, memory=10000, coefficients=10-100, solutionValues=1000, equations=3-10, unknowns=4-15, systems=20, solutions=500, reportSolutions=500
solver: any
Expected outcome:
Solver must not solve such an ANLDE system.

Test case 3.6.1.9 (special). Hard ANLDE system.
Input data:
anlde: the set contains several (one, two, all) large ANLDE system: many solutions, many equations/unknowns, large coefficients
limits: time=180, memory=15000, coefficients=10-1000, solutionValues=10000, equations=3-100, unknowns=6-150, systems=20, solutions=1500, reportSolutions=500

solver: any
Expected outcome:
ANLDE solver must solve these systems; other solvers may fail.

Test case 3.6.1.10 (special). Try to process a large set.
Input data:
anlde: the set is large, test ANLDE systems are simple.
limits: time=180, memory=15000, coefficients=10-1000, solutionValues=10000, equations=3-5, unknowns=6-9, systems=1000-10000, solutions=1500, reportSolutions=500
solver: any
Expected outcome:
Solvers correctly solve them, maybe slowly.

3.7 Class CheckSolutions

Author: Kirill A. Kulakov
Parent subsystem: Algorithm server
Description: Class for comparing Hilbert basises or separating solutions into Hilbert basis.
Class Interfaces: none.

3.7.1 hilbertToHilbert(mainSol, currentSol)

The method comparing Hilbert basises.
Parameters:

1. Solution mainSol: a main Hilbert basis.
   Restrictions: correct Solution object.
2. Solution currentSol: a Hilbert basis which generated by used solving algorithm and which necessary to compare with main Hilbert basis.
   Restrictions: correct Solution object.

Return value: boolean equality: true if basises are equal or false if not equal.
Dependencies: Classes: Solution.
List of tests for a test ANLDE system:

Test case 3.7.1.1 (general). Compare equal Hilbert basises.
Input data:
mainSol: Simple Hilbert basis
currentSol: equal Hilbert basis which can have mixed elements.
Expected outcome:
True.

Test case 3.7.1.2 (negative)**.** Try to compare not equal Hilbert basises.
Input data:
anlde: Hilbert basis
limits: Another Hilbert basis
Expected outcome:
False.

3.7.2 solutionToHilbert(mainSol, currentSol)

The method separate one solution into Hilbert basis.
Parameters:

1. Solution mainSol: a main Hilbert basis.
   Restrictions: correct Solution object.
2. Solution currentSol: a solution which generated by used solving algorithm and which necessary separate into main Hilbert basis.
   Restrictions: correct Solution object.

Return value: boolean equality: true if solution can separate into Hilbert basis or false if it's impossible.
Dependencies: Classes: Solution.
List of tests for a test ANLDE system:

Test case 3.7.2.1 (general)**.** Separate correct solution into Hilbert basis.
Input data:
mainSol: Simple Hilbert basis
currentSol: Some solution which can be separated into main Hilbert basis.
Expected outcome:
True.

Test case 3.7.2.2 (negative)**.** Try to separate wrong solution into Hilbert basis.
Input data:
anlde: Hilbert basis
limits: Some solution which don't can be separated into main Hilbert basis.
Expected outcome:
False.

3.8 Class GeneratorOutputParser

Author: Kirill A. Kulakov.
Parent subsystem: Algorithm server
Description: This class parse ANLDE system from generator output and write it into internal format.
Class Interfaces: None.

3.8.1 parseANLDE()

Translates ANLDE system from the ANLDE System Format the internal format (instance of the ANLDERequest class). IOExceptions are thrown in case of parsing error or when the ANLDE system does not conform to the user limits.
Parameters:

1. String source. ANLDE system in ANLDE System Format.
   Restrictions: None.
2. Boolean set. True if the set is represented, false if one.
   Restrictions: Input par.
3. Limits limits. user limits.
   Restrictions: Input par. Only nonnegative values are used.
4. ANLDE anlde. blank ANLDE object. Need to send signal after end of generating process.
   Restrictions: Output par. The system must satisfy the ANLDE definition. Generator does not generate a ANLDE system when the limits can not be satisfied by this generator or the limits are incorrect.

Return value: This method don't return any value. It filling ANLDE object or call exception ANLDEFormatException.
Dependencies: Classes: ANLDE, Limits, Generator.
List of tests:

Test case 3.8.1.1 (general)**.** Parse a simple generated test (3x6)-ANLDE system.
Input data:
source: see file `./GeneratorParser/out1`.
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=1, solutions=100, reportSolutions=100
anlde: ANLDE object.
Expected outcome:
correct (3x6)-ANLDE system as anlde object.

Test case 3.8.1.2 (positive). Parse a test (5x10)-ANLDE system.
Input data:
source: see file `./GeneratorParser/out2`.
limits: time=100-300, memory=1000-10000, coefficients=5-1000, solutionValues=100-1000, equations=5, unknowns=10, systems=1, solutions=1-1000, reportSolutions=1-500
anlde: ANLDE object.
Expected outcome:
correct ANLDE system as anlde object.

Test case 3.8.1.3 (positive). Parse a test (5x12)-ANLDE system.
Input data:
source: see file `./GeneratorParser/out3`.
limits: time=100-300, memory=1000-10000, coefficients=5-1000, solutionValues=100-1000, equations=5, unknowns=12, systems=1, solutions=1-1000, reportSolutions=1-500
anlde: ANLDE object.
Expected outcome:
correct ANLDE system as anlde object.

Test case 3.8.1.4 (negative). Try to parse a test (1x300)-ANLDE system with incorrect limits.
Input data:
source: see file `./GeneratorParser/out4`.
limits: time=10, memory=1000, coefficients=100, solutionValues=1000, equations=10, unknowns=15, systems=1, solutions=500, reportSolutions=500
anlde: ANLDE object.
Expected outcome:
IOException must be called.

Test case 3.8.1.5 (negative). Try to parse a fail generator output (blank file).
Input data:
source: see file `./GeneratorParser/out51`.
limits: time=10, memory=10000, coefficients=100, solutionValues=1000, equations=3, unknowns=6, systems=10, solutions=100, reportSolutions=100
anlde: ANLDE object.
Expected outcome:
IOException must be called.

Test case 3.8.1.6 (negative). Try to parse a fail generator output (incorrect file).
Input data:
source: see file `./GeneratorParser/out52`.

limits: time=10, memory=10000, coefficients=100, solutionValues=1000, equations=3, unknowns=5, systems=12, solutions=100, reportSolutions=100
anlde: ANLDE object.
Expected outcome:
IOException must be called.

Test case 3.8.1.7 (negative). Try to parse a generator output when number of solution exceed user limit.
Input data:
source: see file `./GeneratorParser/out53`.
limits: time=10, memory=10000, coefficients=100, solutionValues=1000, equations=3, unknowns=100, systems=12, solutions=10, reportSolutions=100
anlde: ANLDE object.
Expected outcome:
IOException must be called.

Test case 3.8.1.8 (special). Try to parse a generator output when process fail because of lack of memory.
Input data:
source: see file `./GeneratorParser/out6`.
limits: time=10, memory=10000, coefficients=100, solutionValues=1000, equations=3, unknowns=6, systems=10, solutions=100, reportSolutions=100
anlde: ANLDE object.
Expected outcome:
IOException must be called.

Test case 3.8.1.9 (special). Try to generate a degenerated test ANLDE system with $n = m = 1$.
Input data:
anlde: null
limits: time=10, memory=10000, coefficients=1000, solutionValues=1000, equations=1, unknowns=1, systems=10, solutions=500, reportSolutions=500
generator: any
Expected outcome:
Depends on the generator used: ANLDE may or may not be generated, but number of equations and unknowns are equal to 1.

Test case 3.8.1.10 (special). Try to generate a square test ANLDE system ($n = m$).
Input data:
anlde: null

limits: time=10, memory=10000, coefficients=1000, solutionValues=1000, equations=unknowns=2-20, systems=10, solutions=500, reportSolutions=500
generator: any
Expected outcome:
Depends on the generator used: ANLDE may or may not be generated, but number of equations and unknowns are equal.

Test case 3.8.1.11 (special). Try to generate a test ANLDE system with one equation only. This results in huge Hilbert basis.
Input data:
anlde: null
limits: time=10, memory=10000, coefficients=100, solutionValues=1000, equations=1, unknowns=2-20, systems=10, solutions=1-1000, reportSolutions=500
generator: any
Expected outcome:
Some ANLDE must not be generated because of exceeding the limit on basis solutions.

Test case 3.8.1.12 (special). Try to generate a test ANLDE system with huge Hilbert basis.
Input data:
anlde: null
limits: time=10, memory=10000, coefficients=100, solutionValues=1000, equations=1-3, unknowns=20-50, systems=10, solutions=1-1000, reportSolutions=500
generator: any
Expected outcome:
Some ANLDE are not generated because of exceeding the limit on basis solutions.

Test case 3.8.1.13 (general). Parse a set of simple test (3x6)-ANLDE systems (20 systems).
Input data:
source: see file `./GeneratorParser/out7`.
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
anlde: ANLDE object.
Expected outcome:
correct set of 20 (3x6)-ANLDE systems as anlde object.

Test case 3.8.1.14 (positive). Parse a set of test (15x20)-ANLDE systems (20 systems).
Input data:
source: see file `./GeneratorParser/out8`.
limits: time=100-300, memory=1000-10000, coefficients=10-1000, solutionValues=100-1000, equations=15, unknowns=20, systems=20, solutions=15-1000, reportSolutions=1-500

anlde: ANLDE object.
Expected outcome:
correct set of 20 (15x20)-ANLDE systems as anlde object.

Test case 3.8.1.15 (special). Parse a set of test (1x20)-ANLDE systems (20 systems).
Input data:
source: see file `./GeneratorParser/out9`.
limits: time=100-300, memory=1000-10000, coefficients=10-1000, solutionValues=100-1000, equations=1, unknowns=20, systems=20, solutions=13-1000, reportSolutions=1-500
anlde: ANLDE object.
Expected outcome:
correct set of 20 (1x20)-ANLDE systems as anlde object.

Test case 3.8.1.16 (negative). Try to parse incorrect set of test (1x20)-ANLDE systems (18 systems instead of 20).
Input data:
source: see file `./GeneratorParser/out10`.
limits: time=100-300, memory=1000-10000, coefficients=10-1000, solutionValues=100-1000, equations=1, unknowns=20, systems=20, solutions=13-1000, reportSolutions=1-500
anlde: ANLDE object.
Expected outcome:
IOException must be called.

3.9 Class BaseSolverOutputParser

Author: Kirill A. Kulakov
Parent subsystem: Algorithm server
Description: This class parse solutions from solver output and write it into internal format.
Class Interfaces: None.

3.9.1 parseOutcome(source, limits, size)

Translates solver outcome to SolverProcess object. SolutionFormatException is thrown in case of parsing error or when the solution does not conform to the user limits. **Parameters:**

1. String source: solution in the solver format.
   Restrictions: None.

2. Limits limits: user limits.
   Restrictions: nonnegative values in object Limits.

3. int size Vertical size of matrix of solutions.
   Restrictions: nonnegative value, the same as vertical size of ANLDE system.

Return value: Filled SolverProcess object or exception SolutionFormatException.
Dependencies: Classes: SolverProcess, SolutionFormatException, Limits, Solver.
List of tests:

Test case 3.9.1.1 (general)**.** Parse typical solution.
Input data:
size: the same as vertical size of ANLDE system. source: see file `./SolverParser/out1`.
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
Expected outcome:
correct solution and solver metrics as SolverProcess object.

Test case 3.9.1.2 (positive)**.** Parse standard solution.
Input data:
size: the same as vertical size of ANLDE system. source: see file `./SolverParser/out2`.
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
correct solution and solver metrics as SolverProcess object.

Test case 3.9.1.3 (special)**.** Parse special (n×n)-ANLDE system solution.
Input data:
size: the same as vertical size of ANLDE system. source: see file `./SolverParser/out3`.
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
Correct solution and solver metrics as SolverProcess object.

Test case 3.9.1.4 (special)**.** Parse special solver result with 0 solution.
Input data:
size: none. source: see file `./SolverParser/out4`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
Correct solution and solver metrics as SolverProcess object.

Test case 3.9.1.5 (negative)**.** Parse special abnormal solver termination.

Input data:
source: see file `./SolverParser/out5`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
IOException must be called.

3.10 Class Lp_solveOutputParser

Author: Kirill A. Kulakov
Parent subsystem: Algorithm server
Description: This class parse solutions from lp_solve solver output and write it into internal format.
Class Interfaces: None.

3.10.1 parseOutcome(source, limits)

Translates solver outcome to SolverProcess object. Two types of exceptions are thrown in case of parsing error or when the solution does not conform to the user limits.
Parameters:

1. String source: solution in the solver format.
   Restrictions: None.

2. Limits limits: user limits.
   Restrictions: nonnegative values in object Limits.

3. int size: number of components in solution.
   Restrictions: None.

Return value: Filled SolverProcess object or exception SolutionFormatException.
Dependencies: Classes: SolverProcess, Limits, Solver.
List of tests:

Test case 3.10.1.1 (general)**.** Parse typical solution.
Input data:
source: see file `./Lp_solveParser/out1`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
size: correct, the same as solving ANLDE system.
Expected outcome:
correct solution and solver metrics as SolverProcess object.

Test case 3.10.1.2 (positive)**.** Parse standard solution.
Input data:
source: see file `./Lp_solveParser/out2`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
size: correct, the same as solving ANLDE system.
Expected outcome:
correct solution and solver metrics as SolverProcess object.

Test case 3.10.1.3 (positive)**.** Parse standard solution.
Input data:
source: see file `./SolverParser/out3`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
size: correct, the same as solving ANLDE system.
Expected outcome:
Correct solution and solver metrics as SolverProcess object.

Test case 3.10.1.4 (special)**.** Parse special outcome with no solutions.
Input data:
source: see file `./SolverParser/out5`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
Correct solution and solver metrics as SolverProcess object.

Test case 3.10.1.5 (negative)**.** Parse special solver outcome with incorrect limits.
Input data:
source: see file `./SolverParser/out4`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
SolutionFormatException must be called.

Test case 3.10.1.6 (negative)**.** Parse special incorrect solver outcome.
Input data:
source: see file `./SolverParser/out6`. limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=20, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:

SolutionFormatException must be called.

3.11 Class UserProfileStore

Author: Andrey V. Ananin.
Parent subsystem: Data store.
Description: Class for reading, writing and deleting user profiles.
Class Interfaces: None.

3.11.1 getUserProfile(nickname)

Getting a user profile (UserProfile object) from the user profile data file. Nickname is used for identify a name of file. Path to the data store required in constructor.
Parameters:

1. String nickname. User's nickname.
   Restrictions: Input par.

Return value: It returns user profile (UserProfile object);
Dependencies:
Classes: UserProfile, Limits.
List of tests:

Test case 3.11.1.1 (general, positive)**.** Get standard user profile from the user profile data file.
Input data:
user profile: null user profile data file:
#User Profile, version 1 group = guest information = Anonymous user. limits.time = 100-300
limits.memory = 1000-10000
limits.coefficients = 5-1000
limits.solutionValues = 100-1000
limits.equations = 2-20
limits.unknowns = 3-20
limits.systems = 1-100
limits.solutions = 1-1000
limits.reportSolutions = 1-500
Expected outcome:
correct user profile as UserProfile object.

Test case 3.11.1.2 (negative)**.** Try to get a user profile from the user profile data file; the file does not exist.

Input data:
user profile: null user profile data file: does not exist
Expected outcome:
The user profile has not been readed: IOException will be thrown.

Test case 3.11.1.3 (negative)**.** Try gets a user profile from the user profile data file, which has wrong format.
Input data:
user profile: null
user profile data file: wrong format
Expected outcome:
The user profile has not been readed: exception will be thrown.

3.11.2 setUserProfile(profile)

Saving user profile UserProfile object) in the data store. Path to the data store required in constructor.
Parameters:

1. UserProfile profile. contains user profile.
   Restrictions: Input par. Limits are nonegative and the other information about a user is a strings.

Return value: It does not return any value;
Dependencies:
Classes: UserProfile.
List of tests:

Test case 3.11.2.1 (general, positive)**.** Save standard user profile (UserProfile object) in the user profile data file.
Input data:
user profile data file: group="guest", information="Anonymous user", limits.time=100-300, limits.memory=1000-10000, limits.coefficients=5-1000, limits.solutionValues=100-1000, limits.equations=2-20, limits.unknowns=3-20, limits.systems=1-100, limits.solutions=1-1000, limits.reportSolutions=1-500
Expected outcome:
correct user profile data file.

3.11.3 removeUserProfile(nickname)

Removing user profile from the data store. Path to the data store required in constructor.
Parameters:

1. String nickname. User's nickname, which identify user profile data file
   Restrictions: Input par.

Return value: It does not return any value;
Dependencies:
Classes: none.
List of tests:

Test case 3.11.3.1 (general, positive)**.** Remove user profile from the data store.
Input data:
user profile data file: exist
Expected outcome:
remove user profile data file.

Test case 3.11.3.2 (negative)**.** Remove user profile from the data store, which can't be deleted.
Input data:
user profile data file: can't be deleted
Expected outcome:
user profile data file has not been deleted: IOException will be thrown.

3.12 Class DefaultLimitsStore

Author: Andrey V. Ananin.
Parent subsystem: Data store.
Description: Class for reading and writing default limits.
Class Interfaces: None.

3.12.1 getDefaultLimits()

Reading default limits (Limits object) from the data store. Path to the data store required in constructor.
Parameters: none.
Return value: It returns default limits as Limits object; It throws exception for any error.
Dependencies:
Classes: Limits.
List of tests:

Test case 3.12.1.1 (general, positive)**.** Gets standard default limits from the default limits file.
Input data:
limits: null default limits file:
# Default limits file, version: 1
time = 100-300
memory = 1000-10000
coefficients = 5-1000
solutionValues = 100-1000
equations = 2-20
unknowns = 3-20
systems = 1-100
solutions = 1-1000
reportSolutions = 1-500
Expected outcome:
correct default limits as Limits object.

Test case 3.12.1.2 (negative)**.** Try gets default limits from the default limits file which does not exist.
Input data:
limits: null default limits file: does not exist
Expected outcome:
The default limits has not been loaded: IOException will be thrown.

Test case 3.12.1.3 (negative)**.** Try gets default limits from the default limits file which has wrong format.
Input data:
limits: null default limits file: wrong format
Expected outcome:
The default limits has not been loaded: exception will be thrown.

3.12.2 setDefaultLimits(limits)

Saving default limits Limits object) in the data store. Path to the data store required in constructor.
Parameters:

1. Limits limits. contains default limits.
   Restrictions: Input par. Only nonnegative values are used.

Return value: It does not return any value;
Dependencies:
Classes: Limits.
List of tests:

Test case 3.12.2.1 (general, positive)**.** Save standard default limits(Limits object) in the default limits file.
Input data:
limits: time=100-300, memory=1000-10000, coefficients=5-1000, solutionValues=100-1000, equations=2-20, unknowns=3-20, systems=1-100, solutions=1-1000, reportSolutions=1-500
Expected outcome:
The correct default limits file.

Test case 3.12.2.2 (negative)**.** Try save default limits(Limits object); A user don't have access to the default limits file.
Input data:
default limits file: access problems
Expected outcome:
The default limits file has not been created: IOexception will be thrown.

3.13 Class StatisticStore

Author: Andrey V. Ananin.
Parent subsystem: Data store.
Description: Class for reading and writing activity statistics.
Class Interfaces: None.

3.13.1 getStatistics()

Reading statistics (list of Statistics objects) from the data store. Path to the data store required in constructor.
Parameters: none.
Return value: It returns list of default limits as Statistics objects;
Dependencies:
Classes: Statistics.
List of tests:

Test case 3.13.1.1 (positive)**.** Get standard activity statistics from the activity statistics log file.
Input data:

activity statistics log file:
guest 0.0.0.0-255.255.255.255 0-10000 0-10000 0-10000 0-10000 0-100 0-10000 0-10000 0-10000 0-2^{32} 0-2^{32}
Expected outcome:
correct list of activity statistics (Statistics object).

Test case 3.13.1.2 (negative)**.** Get standard activity statistics from the activity statistics log file, which does not exist.
Input data:
activity statistics log file: does not exist
Expected outcome:
list of activity statistics (Statistics object) has not been created: IOException will be thrown.

Test case 3.13.1.3 (negative)**.** Get standard activity statistics from the activity statistics log file, which has wrong format.
Input data:
activity statistics log file: has wrong format
Expected outcome:
list of activity statistics (Statistics object) has not been created: exception will be thrown.

Test case 3.13.1.4 (negative)**.** Get standard activity statistics from the activity statistics log file, blank file.
Input data:
activity statistics log file:blank
Expected outcome:
list of activity statistics (Statistics object) has not been created: exception will be thrown.

Test case 3.13.1.5 (special)**.** Get standard activity statistics from the activity statistics log file, only commentary line.
Input data:
activity statistics log file: one commentary line in activity statistics file.
Expected outcome:
list of activity statistics (Statistics object) has not been created: exception will be thrown.

3.13.2 setStatistics(statistics)

Writing statistics (Statistics object) in the data store. Path to the data store required in constructor.
Parameters: none.
Return value: It doesn't return any values.

Dependencies:
Classes: Statistics.
List of tests:

Test case 3.13.2.1 (positive)**.** Add standard activity statistics record to the activity statistics log file.
Input data:
activity statistics log file: nickname="guest", ip="0.0.0.0-255.255.255.255", gensys=0-10000, inputsys=0-10000, solvedsys=0-10000, acknowsys=0-10000, notmatch=0-100, systime=0-10000, worktime=0-10000, memory=0-10000, startsession=0-2^{32}, endsession=0-2^{32}.
Expected outcome:
correctly added activity statistics record in the activity statistics log file.

Test case 3.13.2.2 (negative)**.** Try to add activity statistics record (Statistics object) to the activity statistics log file; input data is wrong.
Input data: wrong input data
Expected outcome:
activity statistics record has not been added: exception will be thrown.

3.14 Class Management

Author: Andrey V. Ananin.
Parent subsystem: Management.
Description: Class for managing user profiles ad default limits, sending notes. This class is testing with "Data store" subsystem integration. **Class Interfaces:** None.

3.14.1 getUserProfile(nickname, password)

Getting a user profile (UserProfile object) from the user profile data file. Nickname is used for identify a name of file. Password and user limits are checked too.
Parameters:

1. String nickname. User's nickname.
   Restrictions: Input par.

2. String password. User's nickname.
   Restrictions: Input par.

Return value: It returns user profile (UserProfile object);
Dependencies:
Classes: UserProfile.
List of tests:

Test case 3.14.1.1 (general, positive)**.** Get standard user profile from the user profile data file.
Input data:
password: correct
user profile: null
correct user profile data file:
#User Profile, version 1
group = guest
password = qwerty
information = Anonymous user. limits.time = 100-300
limits.memory = 1000-10000
limits.coefficients = 5-1000
limits.solutionValues = 100-1000
limits.equations = 2-20
limits.unknowns = 3-20
limits.systems = 1-100
limits.solutions = 1-1000
limits.reportSolutions = 1-500
Expected outcome:
correct user profile as UserProfile object.

Test case 3.14.1.2 (negative)**.** Try to get standard user profile from the user profile data file. Password is wrong.
Input data:
user profile: null
password: wrong
correct user profile data file.
Expected outcome:
Profile data file has not been readed. Exception will be thrown.

Test case 3.14.1.3 (negative)**.** Try to get standard user profile from the user profile data file. User limits are bigger than default limits.
Input data:
user profile: null
user profile data file: user limits are bigger than default limits.
Expected outcome:
Profile data file has been readed. User limits will be replaced by default limits.

3.14.2 getUserProfile(nickname)

Getting a user profile (UserProfile object) from the user profile data file. Nickname is used for identify a name of file. User limits are compared with default limits too.

Parameters:

1. String nickname. User's nickname.
   Restrictions: Input par.

Return value: It returns user profile (UserProfile object);
Dependencies:
Classes: UserProfile.
List of tests:

Test case 3.14.2.1 (general, positive)**.** Get standard user profile from the user profile data file.
Input data:
user profile: null
correct user profile data file:
#User Profile, version 1
group = guest
password = qwerty
information = Anonymous user. limits.time = 100-300
limits.memory = 1000-10000
limits.coefficients = 5-1000
limits.solutionValues = 100-1000
limits.equations = 2-20
limits.unknowns = 3-20
limits.systems = 1-100
limits.solutions = 1-1000
limits.reportSolutions = 1-500
Expected outcome:
correct user profile as UserProfile object.

Test case 3.14.2.2 (negative)**.** Try to get standard user profile from the user profile data file. User limits are bigger than default limits.
Input data:
user profile: null
user profile data file: user limits are bigger than default limits.
Expected outcome:
Profile data file has been readed. User limits will be replaced by default limits.

3.14.3 setUserProfile(userprofile)

Saving a user profile (UserProfile object) in the user profile data file. User limits are compared with default limits .

Parameters:

1. UserProfile userprofile. User profile as UserProfile object.
   Restrictions: Input par.

Return value: It does not return any value.
Dependencies:
Classes: UserProfile.
List of tests:

Test case 3.14.3.1 (general, positive). Save standard user profile (UserProfile object) in the user profile data file.
Input data:
user profile data: group="guest", information="Anonymous user", limits.time=100-300, limits.memory=1000-10000, limits.coefficients=5-1000, limits.solutionValues=100-1000, limits.equations=2-20, limits.unknowns=3-20, limits.systems=1-100, limits.solutions=1-1000, limits.reportSolutions=1-500
Expected outcome:
correct user profile data file.

Test case 3.14.3.2 (negative). Try to save user profile (UserProfile object) in the user profile data file. User limits are bigger than default limits.
Input data:
user profile data: invalid user limits.
Expected outcome:
The user profile data file has been writed; user limits will be replaced by the default limits.

3.14.4 sendNotes(note, anlde, userProfile)

Sending a note to the system administrator.

Parameters:

1. String note. User note.
   Restrictions: Input par. Note format.
2. String anlde. ANLDE system. Restrictions: Input par.
3. String admEmail. Email of system administrator. Restrictions: Input par.

4. UserProfile userprofile. User profile. Restrictions: Input par.

Return value: It does not return any value; It sends email message to the system administator.
Dependencies:
Classes: UserProfile.
List of tests:

Test case 3.14.4.1 (general, positive). Send user note (String and ANLDE system(s) as ANLDE object) to the system administrator.
Input data:
note: user note.
ANLDE: ANLDE system(s).
admEmail: email of the system administrator.
userprofile: User Profile data.
Expected outcome:
Correct note as e-mail message. Send e-mail message to the system administrator.

3.15 Class ActivityStatistics

Author: Andrew Salo
Parent subsystem: Statistics.
Description: This class is used for processing raw statistics data and preparing statistics reports according to requested domain and metric.

3.15.1 getStatisticsReport(domain, metric)

Prepares statistics report according to domain and metric specified. IOException is thrown if an error has been occuried while getting data from statistics store. IllegalArgumentException is thrown if either argument has invalid value.
Parameters:

1. int domain: Integer value specifying domain of statistics request.
   Restrictions: Possible values:
   ActivityStatistics.NICKNAME—requested domain is user nickname.
   ActivityStatistics.IP_ADDRESS—requested domain is ip address.

2. int metric: Integer value specifying metric of statistics request.
   Restrictions: Possible values:
   ActivityStatistics.GENERATED_SYSTEMS—requested metric is number of generated

systems.
ActivityStatistics.INPUT_SYSTEMS—requested metric is number of input systems.
ActivityStatistics.SOLVED_SYSTEMS—requested metric is number of solved systems.
ActivityStatistics.ACKNOWLEDGED_SYSTEMS—requested metric is number of acknowledged systems.
ActivityStatistics.DISCREPANCIES—requested metric is number of solving discrepancies.
ActivityStatistics.USED_SYSTEM_TIME—requested metric is total used system time.
ActivityStatistics.USED_WORK_TIME—requested metric is total used work time.
ActivityStatistics.SESSION_WORK_TIME—requested metric is total session work time.
ActivityStatistics.SESSIONS—requested metric is number of sessions.

Return value:
List of Report objects containing processed statistics data.
Dependencies:
Class StatisticsStore, Statistics, ExtendedStatistics, Report.
List of tests:
The function will be tested during integration testing of ActivityStatistics and DataStore subsystems.

4 Integration Testing

For this project, bottom up integration testing has been chosen as a basic strategy. It is supposed that all components (classes, web pages and servlets) of the web system have been completed and independently tested.

At this phase of testing the components are integrated into clusters, starting with smallest ones and finishing with the whole web system. All these clusters must be tested independently in the order of their apearance.

The following types of tests are for execution:

1. **functional tests:** they test basic or advanced functionality of a subsystem (cluster), may include specific environment conditions or test features of interface during execution of a function.

2. **range tests:** they check out values of parameters and data, which is transferred between subsystems, in different ranges (typical, non-standard, extreme, invalid, etc);

3. **special tests:** they test all important domain specific cases.

Method calls, described in a test case, may result in exception throws due to any special or non-standard events. For exceptions, which are not explicitly described below as a special case, they are assumed to be cathed by the web server, and presented to the end user as a standard error page (automatically generated by the server) with the exception's stack trace. Otherwise, processing of all mentioned exceptions must be tested.

All skeleton clusters of the web system are introduced in the list below. The cluster are listed in the growth order: the next cluster may aggregate the previous ones by adding a new component to a previously constructed cluster.

4.1 Management—DataStore

Integrator(s): A.Anan'in.
Description: The cluster allows to get and set data in the Web-SynDic Data store. The data are user profiles (user information and limits) and default limits. Tests were executed on the unit testing phase.

4.2 ActivityStatistics—DataStore

Integrator(s): A.Salo.
Description: The cluster allows to compute activity statistics metrics for given domain (nickname or IP address). It uses data (log files) from the Data store as primary raw data.

4.2.1 Get raw statistics data from DataStore

This interface is used by ActivityStatistics to get raw statistics data from StatisticsStore.
Input data: Log file containing raw statistics data.
Output data: Statistics report, containing processed statistics data according to requested domain and metric.
Components: ActivityStatistics, StatisticsStore.
List of tests:

Test case 4.2.1.1 (functional)**.** Process typical log file.
Input data: Log file containing raw statistics data.
Expected outcome: Statistics report containing processed statistics data.

Test case 4.2.1.2 (range)**.** Process log file containing only one record.
Input data: Log file containing only one record.
Expected outcome: Statistics report containing processed data.

Test case 4.2.1.3 (range)**.** Process log file containing records with the same domain.
Input data: Log file containing records with the same domain.
Expected outcome: Statistics report containing processed data.

Test case 4.2.1.4 (range)**.** Process log file which has wrong format.
Input data: Log file which has wrong format.
Expected outcome: Exception on wrong log file format.

Test case 4.2.1.5 (special)**.** Process non-existing log file.
Input data: Non-existing log file.
Expected outcome: Exception on not found log file.

4.3 Generator—GeneratorOutputParser

Integrator(s): K.Kulakov, A.Anan'in.
Description: The cluster allows to generate an ANLDE system or a set of them (as ANLDE object). Generator calls an external generator (executable file) and translates its outcome.

4.3.1 Parse ANLDE

Parsing generated ANLDE system or a set of ANLDE systems and build ANLDE object.
Input Data: User limits
Output data: ANLDE object must be created or exception was called.
Components: ANLDE, Limits, GeneratorOutputParser.
List of tests:

Test case 4.3.1.1 (functional)**.** Generating and parsing an ANLDE system.
Input data: User limits.
Expected outcome: ANLDE object must be created.

Test case 4.3.1.2 (functional)**.** Generating and parsing a set of ANLDE system.
Input data: User limits.
Expected outcome: ANLDE object must be created.

Test case 4.3.1.3 (special)**.** Generating and try to parsing an ANLDE system with wrong limits.
Input data:
User limits with small meaning.
Expected outcome: Corect ANLDE system if can be generated or IOException must be called.

Test case 4.3.1.4 (special)**.** Generating and try to parsing a set of ANLDE systems with wrong limits.
Input data:
User limits with small meaning.
Expected outcome: Corect ANLDE system if can be generated or IOException must be called.

Test case 4.3.1.5 (special)**.** Generating and try to parse a large ANLDE system.
Input data:
User limits.
Expected outcome: ANLDE object must be created.

Test case 4.3.1.6 (special)**.** Generating and try to parse a large set of ANLDE systems.
Input data:
User limits.
Expected outcome: ANLDE object must be created.

4.4 Solver—SolverOutputParser

Integrator(s): K.Kulakov, A.Anan'in.
Description: The cluster allows to solve an ANLDE system or a set of them (as ANLDE object). Solver calls an external solver (executable file) and translates its outcome.

4.4.1 parse solver outcome

Parsing solved ANLDE system and build SolverOutcome object.
Input Data: ANLDE object, user limits.
Output data: SolverOutcome object.
Components: ANLDE, Limits, Solver, SolverOutcome, SolverOutputParser, SolverProcess, Solution, FormatException.
List of tests:

Test case 4.4.1.1 (functional)**.** Solving and parsing standart ANLDE system.
Input data: Correct ANLDE object, user limits.
Expected outcome: SolverOutcome object must be created.

Test case 4.4.1.2 (functional)**.** Solving and parsing a set of standart ANLDE systems.
Input data:
Correct ANLDE object, user limits.
Expected outcome: SolverOutcome object must be created.

Test case 4.4.1.3 (special)**.** Try to solve and parse large ANLDE system.
Input data:
Correct ANLDE object, user limits.
Expected outcome: Created SolverOutcome object or called IOException.

Test case 4.4.1.4 (special)**.** Try to solve and parse large set of ANLDE systems.
Input data:
Correct ANLDE object, user limits.
Expected outcome: Created SolverOutcome object or called IOException.

Test case 4.4.1.5 (special)**.** Try to solve and parse ANLDE system with wrong limits.
Input data:
ANLDE object, user limits (small).
Expected outcome: IOException must be called.

Test case 4.4.1.6 (special)**.** Try to solve and parse a set of ANLDE systems with wrong limits.
Input data:
ANLDE object, user limits (small).
Expected outcome: IOException must be called.

4.5 Spooler—Solver—SolverOutputParser—TestSolution

Integrator(s): K.Kulakov.
Description: The cluster allows to solve an ANLDE system or a set of them (as ANLDE object). The solver calls two external solver (executable file, anlde-solver and alternative solver), translates their outcomes and then compares (tests) the solutions trying to uncover discrepancies. This test feature is only for processing one ANLDE system, not a set of them.

4.5.1 Hilbert to Hilbert

Compare two Hilbert basises. Used if alternative algorithm can find Hilbert basis.
Input Data: ANLDE object, user limits, list of alternative algorithms which can find Hilbert basises.
Output data: Field compareResult in class SolverProcess contain true if Hilbert basises are equal, otherwise false.
Components: ANLDE, Limits, Solver, SolverOutcome, SolverOutputParser, SolverProcess, Solution, FormatException, SolverSpooler.
List of tests:

Test case 4.5.1.1 (functional)**.** Compares equal Hilbert basises.
Input data: ANLDE object, user limits, list of alternative algorithms which can find Hilbert basises.
Expected outcome: Field compareResult in class SolverProcess contain true.

Test case 4.5.1.2 (functional)**.** Compares not equal solutions.
Input data: ANLDE object, user limits, list of alternative algorithms which can find Hilbert basises. Alternative algorithms must find another Hilbert basis than main algorithm.
Expected outcome: Field compareResult in class SolverProcess contain false.

4.5.2 Solution to Hilbert

Decomposition solution into Hilbert basis. Used if alternative algorithm can find only one solution.
Input Data: ANLDE object, user limits, list of alternative algorithms which can find only one solution.
Output data: Field compareResult in class SolverProcess contain true if Solution can decompose into Hilbert basis, otherwise false.
Components: ANLDE, Limits, Solver, SolverOutcome, SolverOutputParser, SolverProcess, Solution, FormatException, SolverSpooler.
List of tests:

Test case 4.5.2.1 (functional)**.** Decomposition correct solution into Hilbert basis.
Input data: ANLDE object, user limits, list of alternative algorithms which can find only one solution.
Expected outcome: Field compareResult in class SolverProcess contain true.

Test case 4.5.2.2 (functional)**.** Decomposition uncorrect solution into Hilbert basis.
Input data: ANLDE object, user limits, list of alternative algorithms which can find Hilbert basises. Alternative algorithms must find solution which can not be decompose into Hilbert basis.
Expected outcome: Field compareResult in class SolverProcess contain false.

4.6 Spooler—Generator—*

Integrator(s): K.Kulakov, A.Anan'in.
Description: The cluster allows to serially generate an ANLDE system or a set of them (as ANLDE objects). Spooler contains tasks for generation and in turn calls Generator cluster to execute the next task. Generator calls an external generator (executable file) and translates its outcome.

4.6.1 generateANLDESystem

Generate ANLDE system, run this task and build ANLDE object.
Input Data:
ANLDE: ANLDE system
Limits: User limits
String: Name of generator
Output data: ANLDE object must be filled or exception was called.
Components: ANLDE, Limits, Generator, GeneratorSpooler, GeneratorOutputParser.
List of tests:

Test case 4.6.1.1 (functional)**.** Generate and parse an ANLDE system (one or few tasks).
Input data:
Limits: User limits
String: Name of generator
ANLDE: Not filled ANLDE system.
Expected outcome: ANLDE object(s) must be created.

Test case 4.6.1.2 (range)**.** Generate and parse big ANLDE system (one or few tasks).
Input data:
Limits: User limits for generating big ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Generate and parse big ANLDE system (one or few tasks).

Test case 4.6.1.3 (range)**.** Generate and parse small ANLDE system (one or few tasks); n=m=1;
Input data:
Limits: User limits for generating small ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Generate and parse small ANLDE system (one or few tasks).

Test case 4.6.1.4 (negative)**.** Try to generate and parse an ANLDE system; bad user limits.
Input data:
Limits Not correct user limits
String Name of generator
ANLDe Not filled ANLDE system
Expected outcome: ANLDE.error() method will be called.

Test case 4.6.1.5 (special)**.** Generate and parse square ($n \times n$) ANLDE system.
Input data:
Limits: User limits for generating square ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Generate and parse square ANLDE system (one or few tasks).

Test case 4.6.1.6 (special)**.** Try to generate and parse an ANLDE system, where number of equations is bigger than number of unknowns.
Input data:
Limits: User limits for generating square ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: ANLDE.error() method will be called.

Test case 4.6.1.7 (special)**.** Try to generate and parse an ANLDE system, size of buffer is smaller than number of tasks.
Input data:
Limits: User limits for generating square ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Some tasks will be executed and some other tasks will be rejected (buffer overflow).

4.6.2 generateANLDESystemSet

Generate ANLDE system, run this task and build ANLDE object.
Input Data:
ANLDE: ANLDE system
Limits: User limits
String: Name of generator
Output data: ANLDE object must be filled or exception was called.
Components: ANLDE, Limits, Generator, GeneratorSpooler, GeneratorOutputParser.
List of tests:

Test case 4.6.2.1 (functional)**.** Generate and parse a set of ANLDE systems (one or few tasks).
Input data:
Limits: User limits
String: Name of generator

ANLDE: Not filled ANLDE system.
Expected outcome: Filled set of ANLDE systems.

Test case 4.6.2.2 (range)**.** Generate and parse a set of big ANLDE systems (one or few tasks).
Input data:
Limits: User limits for generating big ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Generate and parse a set of big ANLDE systems (one or few tasks).

Test case 4.6.2.3 (range)**.** Generate and parse a set of small ANLDE systems (one or few tasks); n=m=1;
Input data:
Limits: User limits for generating small ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Generate and parse a set of small ANLDE systems (one or few tasks).

Test case 4.6.2.4 (negative)**.** Try to generate and parse a set of ANLDE systems; bad user limits.
Input data:
Limits: Not correct user limits
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: ANLDE.error() method will be called.

Test case 4.6.2.5 (special)**.** Generate and parse a a set of square ($n \times n$) ANLDE systems.
Input data:
Limits: User limits for generating square ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Generate and parse a set of square ANLDE systems (one or few tasks).

Test case 4.6.2.6 (special)**.** Try to generate and parse a set of ANLDE systems, where number of equations is bigger than number of unknowns.
Input data:
Limits: User limits for generating square ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: ANLDE.error() method will be called.

Test case 4.6.2.7 (special)**.** Try to generate and parse a set of ANLDE systems, size of tasks buffer is smaller than number of tasks.
Input data:
Limits: User limits for generating square ANLDE system
String: Name of generator
ANLDE: Not filled ANLDE system
Expected outcome: Some tasks will be executed and some other tasks will be rejected (buffer overflow).

4.7 SolverSpooler—Solver—*

Integrator(s): K.Kulakov.
Description: The cluster allows to serially solve an ANLDE system or a set of them (as ANLDE objects). Spooler contains tasks for solving and in turn calls Solver cluster to execute the next task.

4.7.1 Solve ANLDE system

This method solving ANLDE system or a set of ANLDE systems and return object SolverOutcome. When SolverSpooler solves current task it calls external Solver object, Solver calls external algorithm and get solver outcome as String object. When Solver object calls SolverOutcomeParser object and gets solver outcome as SolverProcess object. Solver object returns SolverOutcome object. SolverSpooler adds ANLDECh object, Server object and calls method ANLDE.solved().
Input Data: ANLDE object, user limits, list of solvers.
Output data: Called method ANLDE.solved() and SolverOutcome object.
Components: ANLDE, Limits, Solver, SolverSpooler, SolverOutputParser.
List of tests:

Test case 4.7.1.1 (functional)**.** Solving standart ANLDE system using main solver.
Input data: ANLDE system as ANLDE object, user limits, list of solvers contains only main solver.
Expected outcome: Method ANLDE.solved() must be called, SolverOutcome object must be returned.

Test case 4.7.1.2 (functional)**.** Solving set of standart ANLDE systems using main solver.
Input data: A set of ANLDE systems as ANLDE object, user limits, list of solvers contains only main solver.
Expected outcome: Method ANLDE.solved() must be called, SolverOutcome object must be returned.

Test case 4.7.1.3 (functional). Solving standart ANLDE system using set of solvers.
Input data: Standart ANLDE system as ANLDE object, user limits, list of solvers contains a set of solvers.
Expected outcome: Method ANLDE.solved() must be called, SolverOutcome object must be returned.

Test case 4.7.1.4 (functional). Solving a set of standart ANLDE systems using set of solvers.
Input data: A set of standart ANLDE systems as ANLDE object, user limits, list of solvers contains a set of solvers.
Expected outcome: Method ANLDE.solved() must be called, SolverOutcome object must be returned.

4.8 Web server—ANLDE parser

Integrator(s): M.Kryshen and A.Salo.
Description: The cluster allows a user to input ANLDE systems (single or a set) and send them to the web server for further processing. The web server translates the systems into internal format (ANLDERequest object).

4.8.1 Parsing of input ANLDE system in text format

Web server gets ANLDE systems from a user in text format and calls parseANLDE() to convert the system to ANLDERequest object.
Input data: ANLDE system in text format.
Output data: ANLDERequest object.
Components: ANLDE parser, JSP ???.
List of tests:

Test case 4.8.1.1 (functional). Translate a simple (3x6)-test ANLDE system.
Input data:
source: a simple test ANLDE system
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
Expected outcome:
correct ANLDERequest object.

Test case 4.8.1.2 (range). Translate a large (equations, unknowns, coefficients) test ANLDE system.
Input data:
source: a large ANLDE system in the correct format

limits: time=100, memory=3000, coefficients=1000, solutionValues=10000, equations=50, unknowns=100, systems=20, solutions=1000, reportSolutions=1000
Expected outcome:
correct ANLDERequest object.

Test case 4.8.1.3 (functional). Translate a test ANLDE system with double apperance of an unknown in a left-hand side.
Input data:
source: a test ANLDE system with double apperance of an unknown in a left-hand side
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
Expected outcome:
error message containing line number and error description.

Test case 4.8.1.4 (special). Translate a test ANLDE system which contains comments and blank lines.
Input data:
source: a test ANLDE system
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
Expected outcome:
correct ANLDERequest object.

Test case 4.8.1.5 (functional). Translate a set of 3 simple test ANLDE systems.
Input data:
source: a file containing a set of 3 simple test ANLDE systems
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
Expected outcome:
correct ANLDERequest object.

Test case 4.8.1.6 (range). Translate a set of one test ANLDE system.
Input data:
source: a file containing a test ANLDE system
limits: time=100, memory=3000, coefficients=10, solutionValues=100, equations=3, unknowns=6, systems=20, solutions=100, reportSolutions=100
Expected outcome:
correct ANLDERequest object.

Test case 4.8.1.7 (range). Translate a large (systems) set of large (equations, unknowns, coefficients) test ANLDE systems.

Input data:
source: a file containing a large ANLDE system set in the correct format
limits: time=100, memory=3000, coefficients=1000, solutionValues=10000, equations=50, unknowns=100, systems=20, solutions=1000, reportSolutions=1000
Expected outcome:
correct ANLDERequest object.

Test case 4.8.1.8 (functional)**.** Translate a set of test ANLDE systems with systems limit exceeded.
Input data:
source: a file containing a set of test ANLDE systems with systems limit exceeded
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
error message containing line number and error description.

Test case 4.8.1.9 (special)**.** Translate a test ANLDE system set containing systems without equations.
Input data:
source: a file with a set of test ANLDE systems containing systems without equations
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
error message containing line number and error description.

Test case 4.8.1.10 (special)**.** Translate a test ANLDE system set which contains comments and blank lines.
Input data:
source: a file containing a set of test ANLDE systems
limits: time=100, memory=3000, coefficients=50, solutionValues=100, equations=10, unknowns=20, systems=20, solutions=100, reportSolutions=100
Expected outcome:
correct ANLDERequest object.

4.9 Web server—Management—DataStore

Integrator(s): M.Kryshen and A.Ananin.
Description: The cluster allows a registered user to manage her/his data (user profiles). For system administrator it allows to manage any user, including anonymous user, and default limits.

4.9.1 Get User Profile (with password checking)

The Web Server subsystem requests user profile from the data store using Management subsystem when a user logs in (Management subsystem provides password checking).
Input data: User nickname and password.
Output data: User profile (an instance of class UserProfile).
Components: main.jsp, login.jsp, ru.petrsu.websyndic.webserver.Login, ru.petrsu.websyndic.webserver.SessionManager, ru.petrsu.websyndic.management.Management, ru.petrsu.websyndic.management.UserProfile, ru.petrsu.websyndic.datastore.UserProfileStore.
List of tests:

Test case 4.9.1.1 (functional)**.** Input data: request to main.jsp, when no session active.
Expected outcome: main.jsp responds normally, Login form displayed in the login/status area.

Test case 4.9.1.2 (functional)**.** ...
Input data: request to the Login servlet, with valid “nickname” and “password” parameters.
Expected outcome: servlet responds normally with login status information in the login/status area of the page.

Test case 4.9.1.3 (range)**.** Input data: request to the Login servlet, with “nickname” parameter which does not correspond to any registered user..
Expected outcome: servlet responds with “Bad nickname or password” error message.

Test case 4.9.1.4 (range)**.** Input data: request to the Login servlet, with correct “nickname” and wrong “password” parameter.
Expected outcome: servlet responds with “Bad nickname or password” error message.

4.9.2 Get User Profile (no password checking)

The Web Server subsystem requests user profile from the data store using Management subsystem when a user logs in.
Input data: User nickname.
Output data: User profile (an instance of class UserProfile).
Components: userinfo.jsp, limits.jsp, ru.petrsu.websyndic.webserver.UserManager, ru.petrsu.websyndic.webserver.LimtsManager, ru.petrsu.websyndic.management.Management, ru.petrsu.websyndic.management.UserProfile, ru.petrsu.websyndic.datastore.UserProfileStore.
List of tests:

Test case 4.9.2.1 (functional)**.** Input data: request to main.jsp with the “page” parameter set to “userinfo” and “nickname” parameter is valid.
Expected outcome: main.jsp responds normally with the User Information form in the page’s content area.

Test case 4.9.2.2 (functional). Input data: request to main.jsp with the "page" parameter set to "limits" and "nickname" parameter is valid.
Expected outcome: main.jsp responds normally with the User Limits form in the page's content area.

Test case 4.9.2.3 (range). Input data: request to main.jsp with the "page" parameter set to "userinfo" and "nickname" parameter is not valid.
Expected outcome: main.jsp responds with "No such user" error message and the User Management form in the page's content area.

Test case 4.9.2.4 (range). Input data: request to main.jsp with the "page" parameter set to "limits" and "nickname" parameter is not valid.
Expected outcome: main.jsp responds with an Internal Error page.

4.9.3 Save User Profile

The Web Server subsystem invokes the Management's subsystem method to save the specified profile. Management subsystem invokes the profile-saving method from the Data Store subsytem.
Input data: User profile.
Output data: Notification message.
Components: userinfo.jsp, limits.jsp, ru.petrsu.websyndic.webserver.UserManager, ru.petrsu.websyndic.webserver.LimtsManager, ru.petrsu.websyndic.management.Management, ru.petrsu.websyndic.management.UserProfile, ru.petrsu.websyndic.datastore.UserProfileStore.
List of tests:

Test case 4.9.3.1 (functional). Input data: request to the UserManager or LimitsManager servlet (sent by pressing the "Submit" button of the User Limits or User Information form).
Expected outcome: servlet responds normally with a confirmation message, the user profile is saved.

Test case 4.9.3.2 (range). Input data: request to the UserManager or LimitsManager servlet sent by an unauthorized user or request with invalid parameters.
Expected outcome: servlet responds with an error message.

4.9.4 Send Notes

The Web Server subsystem invokes the Management's subsystem method to send user notes to the web system administrator.
Input data: User notes, attached ANLDE system (optional).
Output data: Notification message.

Components: notes.jsp ru.petrsu.websyndic.webserver.NotesDispatcher, ru.petrsu.websyndic.webserver.ANLDERequest, ru.petrsu.websyndic.management.Management.
List of tests:

Test case 4.9.4.1 (functional). Input data: request to the NotesDispatcher servlet to send user notes.
Expected outcome: servlet responds normally with a confirmation message, notes are sent to the administrator.

4.10 WebServer—ActivityStatistics—DataStore

Integrator(s): A.Salo.
Description: The cluster allows the system administrator to select, compute and see various metrics of users activity.

4.10.1 Get statistics report

This interface is used by WebServer to perform a report representing processed statistics data in human-readable form. Raw statistics data is retrieved by StatisticsStore and processed by ActivityStatistics according to requested domain and metric.
Input data: Log file containing raw statistics data.
Output data: Statistics report in Statistics Report Format (described in *Design specification*, section *User interface*).
Components: StatisticsStore, ActivityStatistics, JSP ???.
List of tests:

Test case 4.10.1.1 (functional). Process typical log file.
Input data: Log file containing raw statistics data.
Expected outcome: Statistics report.

Test case 4.10.1.2 (range). Process log file containing only one record.
Input data: Log file containing only one record.
Expected outcome: Statistics report.

Test case 4.10.1.3 (range). Process log file containing records with the same domain.
Input data: Log file containing records with the same domain.
Expected outcome: Statistics report.

Test case 4.10.1.4 (functional). Process log file which has wrong format.
Input data: Log file which has wrong format.
Expected outcome: Exception on wrong log file format.

Test case 4.10.1.5 (special)**.** Process non-existing log file.
Input data: Non-existing log file.
Expected outcome: Exception on not found log file.

Test case 4.10.1.6 (special)**.** Process empty log file.
Input data: Empty log file.
Expected outcome: Exception on corrupted log file.

Test case 4.10.1.7 (special)**.** Process log file containing only one line (treated as comment).
Input data: Log file containing one line.
Expected outcome: Exception on corrupted log file.

4.11 Web server—AlgorithmServer—*

Integrator(s): M.Kryshen and K.Kulakov.
Description: The cluster allows a registered user to input, generate, and solve ANLDE systems. User and default limits are used to control this processing.

4.11.1 get solvers list

Get solvers list from algorithm server.
Input data: none.
Output data: List of solvers name which can be used.
Components: JSP, AlgorithmServer, SolverSpooler.
List of tests:

Test case 4.11.1.1 (functional)**.** get list of solvers, contains same solvers.
Input data: none.
Expected outcome: list of solvers name.

Test case 4.11.1.2 (special)**.** get list of solvers, contains none additional solvers.
Input data: none.
Expected outcome: empty list.

4.11.2 get generators list

Get generators list from algorithm server.
Input data: none.
Output data: List of generators name which can be used.
Components: JSP, AlgorithmServer, GeneratorSpooler.
List of tests:

Test case 4.11.2.1 (functional)**.** get list of generators, contains same generators.
Input data: none.
Expected outcome: list of generators name.

4.11.3 generate ANLDE

Generate one ANLDE system or a set of ANLDE systems.
Input data: user limits, used generator.
Output data: if user want to generate one ANLDE system, then this ANLDE system representing into the corresponding text area. Otherwise, user get message about end of generating process and starts saving and/or solving processes. If generator falture then corresponding message was be represented.
Components: JSP, AlgorithmServer, GeneratorSpooler.
List of tests:

Test case 4.11.3.1 (functional)**.** generate an ANLDE system.
Input data: user limits, used generator.
Expected outcome: an ANLDE system must be generated and represented into corresponding text area. If generating process was falture then corresponding message was be represented.

Test case 4.11.3.2 (functional)**.** generate a set of ANLDE systems.
Input data: user limits, used generator. For simple execution checkbox "Save generated set" may be checked.
Expected outcome: a set of ANLDE systems must be generated and corresponding message was be represented. If generating process was falture then corresponding message was be represented.

4.11.4 solve ANLDE

Solve one ANLDE system or a set of ANLDE systems.
Input data: ANLDE system or a set of ANLDE systems, user limits, used additional solver.
Output data: If user solve one ANLDE system, then solver outcome was be represented into "ANLDE System Solution Format", otherwise if user solve a set of ANLDE systems she/hi get solver outcome into "ANLDE System Set Solution Format". If solving process was falture then corresponding message was be represented.
Components: JSP, AlgorithmServer, GeneratorSpooler.
List of tests:

Test case 4.11.4.1 (functional)**.** solve an ANLDE system.
Input data: ANLDE system, user limits, used additional solver.

Expected outcome: User gets solver outcome into the “ANLDE System Solution Format”. If solving process was falture then corresponding message was be represented.

Test case 4.11.4.2 (functional)**.** solve a set of ANLDE systems.
Input data: a set of ANLDE systems, user limits, used additional solver.
Expected outcome: User gets solver outcome into the “ANLDE System Set Solution Format”. If solving process was falture then corresponding message was be represented.

5 Validation Testing

The testing strategy was described in Section 2.3. Validation/acceptance testing is an essential part of the testing phase. The web system is tested as a result product of the project in accordance with Validation Criteria. This testing is use case based and ensures that all features of the web system meet their requirements.

It is assumed, that input data of a user is correct in context, otherwise an error message is shown in Content area, along with the form with the values previously entered by the user.

The rest of the section is a list of features to test; each feature corresponds to a certain use case. The only exception is performance issues, which are related to several use cases.

5.1 Session

User session management is mainly based on Java internal features. Thus, the high-level functionality is the only issue to test here.
Description: A user starts a session, make required processing, and then she/he finishes the session.
Type of test: functional, acceptance, security.
Testing:

1. Start Session

   User Actions: User opens a Web-SynDic page with her/his Internet browser.

   Expected Outcome: The required Web-SynDic page is presented to the user, a Log In form appeares in this page.

2. Work During Session

   Precondition: Session is started.

   User Actions: User clicks on any link/button in a current Web-SynDic page, or just reloads page content.

   Expected Outcome The required Web-SynDic page is presented to the user. In case of disabled browser cookies, URL fields contains the user session ID.

3. Session Timeout

   Precondition: Session is started. User has done nothing for a fixed period of time (the timeout is set by sysadmin with configuration files, see tomcat features, we choose 15 minutes as default).

   User Actions: User clicks on any link/button in a current Web-SynDic page or reload the content.

   Expected Outcome: For the registered user, the result is the same as Log Out (see 5.4) test case has been performed. The Start Session test case is started. All user data within the previous session are lost, if the data were not saved in the Data store. For anonymous user all new data are lost.

4. Session Interrupt

   Precondition Session is started for a registered user.

   User Actions: User interrupts HTTPconnection with the web system (e.g. a network connection interrupt, browser closing, etc). After some time, less than the timeout value, the user reestablishes HTTP connection, and continues to work with the web system.

   Expected Outcome: Session successfully continues:

   (a) all user data within the session remain accessible;
   (b) in case of disabled browser cookies, the user session ID remains unchanged;
   (c) status information for registered user remains;
   (d) all special links for registered user are accessible.

5. Simultaneous Browser Windows

   Precondition: Session is started.

   User Actions: User opens one or more browser windows, loads any Web-SynDic page within these windows, and begins to work with the web system using opened browser windows.

   Expected Outcome: For this case, in different windows different sessions have been started. Each session has its own anlde object (ANLDE system or a set of them) and other sessions have not influence on it. All sessions with the same nickname share the same user profile; a change in one session results in the corresponding change in all other sessions. Exception is the anonymous user—each session has its own profile and the changes have no effect to other sessions of anonymous.

6. Simultaneous Processings

   Precondition: Several sessions have been started by several users (different or the same).

   User Actions: Each user in her/his browser window executes Process a set of ANLDE systems and/or Process an ANLDE system use cases concurrently with the other sessions.

   Expected Outcome ANLDE objects are pushed into the queue and processed sequantially. If the queue is full, the error message is presented to a user within the corresponding session.

7. Same session ID

   Precondition: Session has been started by user.

   User Actions: A user know session ID and copy it in new browser window.

   Expected Outcome The same session open. A user continue work with all browser windows as in one browser window. If user limits, user profile data are changed in one browser window, all changes will be submitted immediately and if the user change this data in other browser window it will be have new values. In case of trying to process more than one ANLDE system or set of ANLDE systems error message will be shown (can not process more than one ANLDE system or more than one set of ANLDE systems). If a user logout from the web-system in one browser window, all attempts to submit changes in session in other brower windows will be failed - session will be closed.

References: EUR: EU0: User session starting and finishing, SR: startSession, SR: finishSession, UC: Work with Web-SynDic, UI: Style Convention: Page Layout, UI: IO Data Formats: Acknowledgment Format, UI: IO Data Formats: Error Message Format.

5.2 Process an ANLDE system

Description: The web system solves a test ANLDE system and produces the report on solution.
Type of test: functional, acceptance.
Precondition: One of the Web-SynDic system pages is opened in user's browser.
Testing:

1. UI process form

   User Actions: User clicks on Process an ANLDE System link in the Main Menu.

   Expected Outcome: The user is returned with Process an ANLDE System form in the Content area.

2. ANLDE inputting

   User Actions:

   Generation: User clicks on Generate button of Process an ANLDE System form.

   Input: User tries to manually input an ANLDE System of Process an ANLDE System form.

   Expected Outcome: the input/generated ANLDE system is appearing in the text area of Process an ANLDE System form.

3. Saving

   User Actions: User clicks on Save button of Process an ANLDE System form.

   Expected Outcome: The user is returned with the new browser windows containing textual presentation of the ANLDE system; it is shown in System field of Process an ANLDE System form.

4. Solving

   User Actions:

   Standard: User clicks on Solve button of Process an ANLDE System form.

   Comparing: User select slopes , or any other algorithm corresponding to one of external ANLDE solvers, and clicks on Solve button of Process an ANLDE System form.

   Expected Outcome: If processing of the ANLDE system takes more than 20 seconds, user is returned with Progress Message every 20 seconds until processing is done.

   When processing is done, the user is returned with the report on solution of the ANLDE system, for the ANLDE syntactic algorithm and for any used external solver.

Example test data:

User Input:

```
x1 + x11            = 82*x1 + 56*x2 + 48*x5 + 22*x8 +
                      19*x10 + 49*x11 + 92*x12
x2 + x7 + x9 + x12 = 50*x1 + 48*x2 + 18*x5 + x7 + 20*x8 +
                      x9 + 86*x10 + 71*x11 + 30*x12
x3 + x10 + x14      = 15*x1 + 66*x2 + x3 + 83*x5 + 76*x8 +
                      74*x10 + 96*x11 + 62*x12 + x14
x4 + x8             = 29*x1 + 29*x2 + x4 + 28*x5 + 86*x8 +
                      36*x10 + 74*x11 + 5*x12
x5 + x6 + x13       = 51*x1 + 26*x2 + 3*x5 +x6 + 33*x8 +
                      81*x10 + 49*x11 + 54*x12 + x13
```

Expected Solution:

1. Test ANLDE system:

```
x1 + x11           = 82*x1 + 56*x2 + 48*x5 + 22*x8 +
                     19*x10 + 49*x11 + 92*x12
x2 + x7 + x9 + x12 = 50*x1 + 48*x2 + 18*x5 + x7 + 20*x8 +
                     x9 + 86*x10 + 71*x11 + 30*x12
x3 + x10 + x14     = 15*x1 + 66*x2 + x3 + 83*x5 + 76*x8 +
                     74*x10 + 96*x11 + 62*x12 + x14
x4 + x8            = 29*x1 + 29*x2 + x4 + 28*x5 + 86*x8 +
                     36*x10 + 74*x11 + 5*x12
x5 + x6 + x13      = 51*x1 + 26*x2 + 3*x5 +x6 + 33*x8 +
                     81*x10 + 49*x11 + 54*x12 + x13
Number of solutions: 7
```

2. Performance metrics of solvers:

Algorithm	System time(sec)	Work time(sec)	Memory usage(Kb)	Solving result
anlde	0.00	0.164492	2192	Solved
slopes	0.72	2.764164	2192	abnormal solver terminatic

3. Processing machine characteristics:
   - **CPU:** IA32, 1200 MHz;
   - **RAM:** 256 MB.
   - **Operating system:** Linux 2.4.19
   - **Priority(nice):** 15.

4. Solutions of testing ANLDE system for each algorithm:

```
        x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14
h1      0  0  1  0  0  0  0  0  0  0   0   0   0   0
h2      0  0  0  1  0  0  0  0  0  0   0   0   0   0
h3      0  0  0  0  0  1  0  0  0  0   0   0   0   0
h4      0  0  0  0  0  0  1  0  0  0   0   0   0   0
h5      0  0  0  0  0  0  0  0  1  0   0   0   0   0
h6      0  0  0  0  0  0  0  0  0  0   0   0   1   0
h7      0  0  0  0  0  0  0  0  0  0   0   0   0   1

slopes: cannot cope with the task.
```

References: EUR: EU1a: Solving a test ANLDE system, SR: solveANLDESystem, SR: inputANLDESystem, SR: generateANLDESystem, SR: saveANLDESystems, SR: sendANLDESystem, SR: sendANLDESystemReport, SR: sendAcknowledgments, SR: sendProcessMessage, UC: Process an ANLDE system, UI: Style Convention: Page Layout, UI: Forms: Process an ANLDE System, UI: IO Data Formats: ANLDE System Format, UI: IO Data Formats: ANLDE System Solution Format, UI: IO Data Formats: Process Message Format, UI: IO Data Formats: Error Message Format.

5.3 Process a set of ANLDE systems

Description: The web system solves set of ANLDE systems and produces the report on solution.
Type of test: functional, acceptance.
Precondition: One of the Web-SynDic system pages is opened in the user's browser.
Testing:

1. UI process form

   User Actions: The user clicks on Process a Set of ANLDE Systems link in Main Menu.

   Expected Outcome: The user is returned with Process a Set of ANLDE Systems form in the Content area.

2. ANLDE systems set inputting

   User Actions:

   Generation: User can set Generate a new set radio button, Solve generated set or Save generated set in Process an ANLDE System form.

   Input: User can set the Load set from a text file and solve radio button and manually enters path to text file containing a set of ANLDE Systems to solve in text area, or clicks on Browse button and chooses file in the appeared browse window.

   Expected Outcome The corresponding forms and radio buttons are filled properly.

3. Solving

   User Actions:

   Standard: the user clicks on Process button of Process a Set of ANLDE System form.

   Comparing: the user sets slopes checkbox, or any other checkbox corresponding to one of external ANLDE solvers, and clicks on Process button of Process a Set of ANLDE System form.

Expected Outcome: If generation or processing of the set of ANLDE systems takes more than 20 seconds, user is returned with Progress Message every 20 seconds until processing is done.

When processing is done, the user is returned with following results, depending on set parameters.

Set	**Result**
Load set from a text file and solve radio button	report on solution
Generate a new set radio button and Solve generated set checkbox	report on solution
slopes checkbox, and any of two previous cases	report on solution: internal algorithms and external solvers
Generate a new set radio button and Save generated set checkbox	new browser window with the generated set of ANLDE systems

Example test data:

User Input

```
x1 + x4 = 2*x1 + 3*x3
x2 + x3 = x1 + 2*x2 + x3
%
x1 + x11           = 82*x1 + 56*x2 + 48*x5 + 22*x8 +
                     19*x10 + 49*x11 + 92*x12
x2 + x7 + x9 + x12 = 50*x1 + 48*x2 + 18*x5 + x7 + 20*x8 +
                     x9 + 86*x10 + 71*x11 + 30*x12
x3 + x10 + x14     = 15*x1 + 66*x2 + x3 + 83*x5 + 76*x8 +
                     74*x10 + 96*x11 + 62*x12 + x14
x4 + x8            = 29*x1 + 29*x2 + x4 + 28*x5 + 86*x8 +
                     36*x10 + 74*x11 + 5*x12
x5 + x6 + x13      = 51*x1 + 26*x2 + 3*x5 +x6 + 33*x8 +
                     81*x10 + 49*x11 + 54*x12 + x13
%
x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 +
x9 + x10 + x11 + x12 + x13 + x14
                   = 90*x1 + x2 + 7*x3 + 88*x4 + 96*x5 +
                     59*x6 + x7 + 7*x8 + x9 + x10 + 63*x11 +
                     55*x12 + x13 + 89*x14
```

```
%
x1      = x1
x2 + x3 = x2 + x3
```

Expected Solution

1. ANLDE system metrics:

```
minimum number of equations: 1
average number of equations: 2.5
maximum number of equations: 5

minimum number of unknowns: 3
average number of unknowns: 8.75
maximum number of unknowns: 14

maximum coefficients in ANLDE systems: 96

minimum number of solutions: 1
average number of solutions: 4
maximum number of solutions: 7
```

2. Algorithm metrics:

Algorithm	System time(sec)	Work time(sec)	Memory usage(Kb)	Solving result
anlde	0.00	1.513098	2192	Solved
slopes	0.72	3.483165	2192	Abnormal solver terminati

3. Solving machine characteristics:
   - **CPU:** IA32, 1200 MHz;
   - **RAM:** 256 MB.
   - **Operating system:** Linux 2.4.19
   - **Priority(nice):** 15.

References: EUR: EU1a: Solving a test ANLDE system, SR: solveANLDESystemSet, SR: generateANLDESystemSet, SR: loadANLDESystems, SR: saveANLDESystems, SR: sendANLDESystemSet, SR: sendANLDESystemSetReport, SR: sendAcknowledgments, SR: sendProcessMessage, UC: Process a set of ANLDE systems, UI: Style Convention: Page Layout, UI: Forms: Process a set of ANLDE systems, UI: IO Data Formats: ANLDE System Set Format, UI: IO Data Formats: ANLDE System Set Solution Format, UI: IO Data Formats: Process Message Format, UI: IO Data Formats: Error Message Format.

5.4 Log In / Log Out

Description: The web system allows user to log in.
Type of test: functional, acceptance.
Precondition: One of the Web-SynDic system pages is opened in user's browser.
Testing:

1. Log In

   Precondition: The user has not yet logged in.

   User Actions: The user fills Login and Password fields in the Log In form of the current Web-SynDic page, and clicks on Log in button.

   Expected Outcome: The start Web-SynDic page appears with:

   (a) area of Log In form contains status of the user:

   `You have logged in as <nickname> (log out).`

   where `log out` is a link for the user to log out.

   (b) links to registered user information (user profile and limits) appear in User Menu;

   (c) in case of the sysadmin login: administration links (user profile, user limits, manage users, bounds on limits, and activity statistics) appear in User Menu.

2. Log Out

   Precondition: The user has logged in.

   User Actions: The user clicks on the Log out link in the area of Log In form.

   Expected Outcome: Start Web-SynDic page is loaded with Log in form. The previous session has been finished, the session for anonymous user is started.

References: EUR: EU2d: Log In, SR: logInUser, SR: sendAcknowledgments, UC: Log In, UI: Style Convention: Page Layout, UI: Forms: Log In, UI: IO Data Formats: Acknowledgment Format, UI: IO Data Formats: Error Message Format.

5.5 Send a note

Description: The web system allows user to send her/his opinion on the solution result. A special case here is user's disagreement with found solution(s) of the processed ANLDE system.
Type of test: Functional, acceptance.
Precondition:

1. For test case **Notes on Solution**, one of ANLDE systems has already been solved by the Web-SynDic system, and page with the solution of the last one is opened in the user's browser.
   For test case **General Notes** this is not required.

2. One of the Web-SynDic system pages is opened in the user's browser.

Testing:

1. General Notes

   (a) User Actions: The user clicks on Notes link in Main Menu.

   Expected Outcome: The user is returned with General Notes form in the Content area.

   (b) User Actions: The user feed in his/her textual comment in the text area and clicks on OK button in General Notes form.

   Expected Outcome: The user is returned with acknowledgment about successfull delivery in the Content area.

2. Notes on Solution

   (a) User Actions: The user fills fields and clicks on the OK button of in Notes on Solution form in Content area. (The form is included in the report on solution of solved ANLDE system or set of systems.)

   Expected Outcome: The user is returned with acknowledgment about successfull delivery in the Content area.

References: EUR: EU2b: User notes, SR: sendUserNotes, SR: sendAcknowledgments, UC: Send a note, UI: Style Convention: Page Layout, UI: Forms: General Notes, UI: Forms: Notes on Solution, UI: IO Data Formats: Note Format, UI: IO Data Formats: Acknowledgment Format.

5.6 Register a User

Description: The web system allows user to register when she/he wishes. The registered user has an unique identifier (nick name).
Type of test: functional, acceptance.
Precondition: One of the Web-SynDic system pages is opened in the user's browser.
Testing:

1. User Actions: The user clicks on Register button in the Log In form.

   Expected Outcome: Registration form is loaded in Content area.

2. User Actions: The user feed in required fields of provided Registration form, and, may be, not obligatory field(s), afterward clicks on Register button.

   Expected Outcome: The user is returned with acknowledgment message about successfull registration in Content area.

References: EUR: EU2a: User registration, SR: registerUser, SR: sendAcknowledgments, UC: Register a user, UI: Style Convention: Page Layout, UI: Forms: Log In, UI: Forms: Registration, UI: IO Data Formats: Acknowledgment Format, UI: IO Data Formats: Error Message Format.

5.7 Manage user limits

Description: A regular user may manage her/his own limits on the solution process.
Type of test: Functional, acceptance.
Precondition: One of the Web-SynDic system pages is opened in the user's browser.
Testing:

1. User Actions:

   Standard: the user clicks on the Manage Limits link in User Menu.

   Second: the user clicks on the Change Limits link of Process an ANLDE System form or Process a Set of ANLDE Systems form.

   Expected Outcome: The user is returned with the User Limits form in the Content area, with:

   (a) if the user is registered, fields are filled with values, written in his/her user profile (default, if the user has not set them before);

   (b) otherwise, fields are filled with values, stored in current session limits (default, if the user has not set them during current session).

   Besides, corresponding value of default limit to each field of the form are presented.

2. User Actions: The user edits values of fields, and clicks on Submit new values button in User Limits form.

   Expected Outcome: The user is returned about successfull changes in Content area.

References: EUR: EU2c: Regular User Limits Management, SR: manageUserLimits, SR: sendAcknowledgments, UC: Manage user limits, UI: Style Convention: Page Layout, UI: Forms: User Limits, UI: IO Data Formats: Limits Format, UI: IO Data Formats: Acknowledgment Format, UI: IO Data Formats: Error Message Format.

5.8 Manage Users

Description: The web system allows administrator to change or remove user accounts.
Type of test: Functional, acceptance.
Precondition:

1. User is logged as system administrator.
2. One of the Web-SynDic system pages is opened in the user's browser.

Testing:

1. User Actions: The user clicks on the Manage Users administrative link in User Menu.

   Expected Outcome: The user is returned with the Manage Users form in the Content area.

2. User Actions: The user feeds in Nickname fields, and clicks on Edit button in Manage Users form.

   Expected Outcome: The user is returned with the User Information form in the Content area.

3. User Actions: The user feeds in fields, and clicks on Submit new values button in User Information form.

   Expected Outcome: The user is returned about successfull account changes (if Remove account option was not set), or account removal (if Remove account option was set), in Content area.

References: EUR: EU3a: User Management, SR: manageUsers, SR: sendAcknowledgments, UC: Manage Users, UI: Style Convention: Page Layout, UI: Forms: Manage Users, UI: Forms: User Information, UI: IO Data Formats: User Information Format, UI: IO Data Formats: Acknowledgment Format, UI: IO Data Formats: Error Message Format.

5.9 Manage default limits

Description: The web system allows administrator to change default limits on solution process.
Type of test: Functional, validation.
Precondition:

1. User is logged as system administrator.
2. One of the Web-SynDic system pages is opened in the user's browser.

Testing:

1. User Actions: The user clicks on the Manage Default Limits link in User Menu.

   Expected Outcome: The user is returned with the Default Limits form in the Content area, filled with current default limits.

2. User Actions: The user edits values of fields, and clicks on Submit new values button in Default Limits form.

   Expected Outcome: The user is returned about successfull changes in Content area.

References: EUR: EU3c: Default Limits Management, SR: manageDefaultLimits, SR: sendAcknowledgments, UC: Manage default limits, UI: Style Convention: Page Layout, UI: Forms: Default Limits, UI: IO Data Formats: Limits Format, UI: IO Data Formats: Acknowledgment Format, UI: IO Data Formats: Error Message Format.

5.10 Get Statistics

Description: The web system computes user activity statistics (available for administrator only). The web system generates report on generated and solved ANLDE systems, resource consumption and user notes statistics.
Type of test: Functional, acceptance.
Precondition:

1. User is logged as system administrator.
2. One of the Web-SynDic system pages is opened in the user's browser.

Testing:

1. User Actions: The user clicks on the Activity statistics administrative link in User Menu.

   Expected Outcome: The user is returned with the Activity Statistics form in the Content area.

2. User Actions: The user chooses Domain and Metric fields, and clicks on Get report button in Activity Statistics form.

   Expected Outcome: If activity statistics processing takes more than 20 seconds, user is returned with Progress Message every 20 seconds until processing is done.

   When processing is done, the user is returned with the Activity Statistics report in the Content area.

Example test data:

1. IP address domain.

   User Input:

```
guest 192.168.1.1 15 13 14 1 1 0.1  9.00  2192
guest 192.168.1.1 16 16 15 1 0 0.11 10.10 3192
user2 192.168.2.2 17 17 16 2 0 0.12 20.20 4192
guest 192.168.2.2 18 18 17 1 0 0.13 30.30 5192
user2 192.168.1.1 19 19 18 5 0 0.14 40.40 6192
user3 192.168.1.1 20 20 19 4 1 0.15 50.50 7192
user2 192.168.2.2 21 21 20 3 1 0.16 61.00 8192
user2 192.168.2.2 22 20 21 1 0 0.17 71.10 9192
user2 192.168.1.1 21 19 20 3 0 0.16 61.00 8192
guest 192.168.1.1 20 18 19 1 0 0.15 50.50 7192
guest 192.168.3.3 19 17 18 1 1 0.14 40.40 6192
user2 192.168.3.3 18 16 17 0 2 0.13 30.30 5192
user3 192.168.3.3 17 15 16 0 2 0.12 20.20 4192
guest 192.168.1.1 16 14 15 0 0 0.11 10.10 3192
user3 192.168.2.2 15 13 14 0 1 0.1  9.0   2192
```

   User Actions: The user chooses IP address as Domain value, and Solved systems as Metric value.

   Expected Outcome:

```
192.168.1.1 120
192.168.2.2 88
192.168.3.3 51
```

2. Nickname domain.

   User Input:

```
guest 192.168.1.1 15 13 14 1 1 0.1  9.00  2192
guest 192.168.1.1 16 16 15 1 0 0.11 10.10 3192
user2 192.168.2.2 17 17 16 2 0 0.12 20.20 4192
guest 192.168.2.2 18 18 17 1 0 0.13 30.30 5192
user2 192.168.1.1 19 19 18 5 0 0.14 40.40 6192
user3 192.168.1.1 20 20 19 4 1 0.15 50.50 7192
```

```
user2 192.168.2.2 21 21 20 3 1 0.16 61.00 8192
user2 192.168.2.2 22 20 21 1 0 0.17 71.10 9192
user2 192.168.1.1 21 19 20 3 0 0.16 61.00 8192
guest 192.168.1.1 20 18 19 1 0 0.15 50.50 7192
guest 192.168.3.3 19 17 18 1 1 0.14 40.40 6192
user2 192.168.3.3 18 16 17 0 2 0.13 30.30 5192
user3 192.168.3.3 17 15 16 0 2 0.12 20.20 4192
guest 192.168.1.1 16 14 15 0 0 0.11 10.10 3192
user3 192.168.2.2 15 13 14 0 1 0.1  9.0   2192
```

User Actions: The user chooses Nickname as Domain value, and Acknowledged systems as Metric value.

Expected Outcome:

```
guest 5
user2 14
user3 4
```

References: EUR: EU3b: Activity Statistics, SR: requestStatisticsReport, SR: updateStatistics, UC: Get statistics, UI: Style Convention: Page Layout, UI: Forms: Activity Statistics, UI: IO Data Formats: Statistics Report Format, UI: IO Data Formats: Activity Domain List Format, UI: IO Data Formats: Activity Metrics List Format, UI: IO Data Formats: Process Message Format.

5.11 Performance Tests

Description: According to User Requirements: Performance,

1. AS1: the web system must serve concurrently up to 5 users (separate user sessions) without significant reduction of the server performance;
2. AS2: the web system must not overload a base server more than 75% of the total server workload;
3. AS3: the web system must reply on a user action less than after 20 seconds.

Note, that Req. AS3 is tested in Process an ANLDE system (see. 5.2), Process a set of ANLDE systems (see. 5.3) and Get Statistics (see. 5.10) test specifications, and will not be specially mentioned in this section.

It is supposed that both the AS1 and AS2 Requirements will be tested simultaneously (while performing test case).

Type of test: Performance, acceptance.

Precondition:

1. No less than 5 users must participate in the test execution.
2. Each participated user (hereafter: the users) uses the only browser.
3. All the ANLDE systems and set of systems must be prepared before the actual test execution, to minimize waste of time for data input.
4. One of the Web-SynDic system pages is opened in the each browser.

Testing:

User Actions: The users simultaneously performs Process an ANLDE system and Process a set of ANLDE systems test cases.

Server Control: Permanent control over server metrics: general server workload, memory consumption, etc.

Expected Outcome: On server: according to the Performance Requirements AS1 and AS2.

References: UR: Performance, SR: sendProcessMessage, UI: IO Data Formats: Process Message Format.