Software Engineering: System Test

1. Introduction

These lecture notes describe typical system tests for academic/research hydroinformatics software components for numerical simulation. System tests for commercial software systems includes much more complex and different test cases, scenarios and strategies and are not target of these lecture notes.

The system tests can be subdivided in tasks:

- formal implementation check
- analytic test cases
- performance test
- user interface test
- system component test
- case studies

Systems tests on operability on different platforms, hardware architectures and so on will be not considered in these lecture notes. Tests of the numerical scheme such as stability, osscillation, error propagation etc. are also not handled here.

2. Formal Implementation Check

The formal implementation check is mainly supported by programming tools such as compiler and linker and includes the syntax check of the code as well as the completeness of the system environment by other class packages and libraries.

In Java the Java compiler javac transforms the Java code in virtual machine code checking the code syntax. Errors related to the syntax of the programming language are listed to the software developer by the compiler.

The syntax check of the compiler can only test the correct use of the programming language specified by the programming language syntax and rules. Semantic programming “errors” can not been identified. Example is a for-Loop in Java with correct syntax but with a “wrong” semi-colon after the `for` statement:

```
for (i=0;i<number;i++) ;
  vector[i] = 2.* load.getValue(i);
```

The completeness of the machine code is checked by the linker tools for compiler based programming languages. As Java is using a mixed compiler/interpreter approach this task is done by the compiler and during the execution in the Java virtual machine. Generating the virtual machine code the compiler is looking for the required class links within the CLASSPATH environment. Missing class files or class method will be compiled from source or viewed by compiler messages. Similar is done during execution of virtual machine code within a Java engine / Runtime environment. Missing classes will be viewed by related Java exceptions to the runtime environment.

3. Analytic Test Cases

The analytic test cases are set-up to check the basic numerical/functionality of the specific hydroinformatics system and to identify existing problems within the source code of the related numerical simulation component. Key idea is to specify simple test cases with known results by analytic solutions or physical inside and with a reduced number of relevant parameters with impact on the results.

The impact of a system parameter to the numerical simulation results can be eliminated by using standard, normalized values such as 0 or 1. Parameter multiplied to equation terms such as material constants can be set to 1 in the whole model domain to eliminate their impact on the numerical solution. Parameter added to equation terms such as loads/source terms in the domain or at the boundary can be set to 0 to eliminate their impact on the numerical solution. The geometry can be simplified by regular structures and same, normalized properties in all dimensions. Physical behaviour can be reduced to phenomena in lower dimension, e.g. checking 1D behaviour for a 2D system.

3.1 Example FDM 2D

This section will specify some basic test cases for a groundwater 2D FDM system for rectangle shaped model domain with global hydraulic conductivity values. Parameters to specify the test cases are:

- N_1,N_2 number of nodes in direction 1 and 2
- L_1,L_2 length of model domain in direction 1 and 2
- K_{11},K_{22} hydraulic conductivity in direction 1 and 2
- $w(x_1,x_2)$ source/sink term in the model domain
- BC North boundary condition on the north side, h or q with given value
- BC South boundary condition on the south side, h or q with given value
- BC East boundary condition on the east side, h or q with given value
- BC West boundary condition on the west side, h or q with given value

The combination of all parameter and related suitable value sets leads to an enormous number of test cases. Not all of them have tcarry out. An intelligent selection will lead to an efficient analytic check and problem identification in case of failed tests. Examples for basic analytic test cases will be explained.

Test case 1 is checking the impact of the node number. All other parameters are set to 0 or 1, the boundary conditions to a fixed head of 0. This leads to a constant head field $h(x_1,x_2) = 0$ without any flow. The minimum number of nodes is 3 (one inner node). The number can be varied for larger odd and even node number. The variation of the node number will be done for both directions in parallel or independent.

Test case 2 is checking the impact of the system length. The variation of the length in both directions is done in combination with different node numbers to test typical space approximation combinations.

Test case 3 is checking the impact of the hydraulic conductivity. Similar to test case 2 the K_{ii} values will be used in several variations of same and different values. Values equal 0 will lead to a singular matrix. Setting the value of one direction to a “very small”

in comparison to the other direction means physically to prefer flow in the direction with the higher conductivity. This leads to the system behaviour of a 1D system.

Test case 4 will check the impact of the boundary conditions. A fixed head value at all boundaries will lead to a constant head field without flow. The change of one, two or three boundary conditions to q=0 should not change this result. For four boundary conditions q=0 the numerical system should lead to a singular matrix.

Test case 5 checks the impact of the head boundary conditions. For all four boundary conditions the head values are set to different values. These values along the boundaries have to be part of the final result.

Test case 6 is using a 1D analytic solution to test the system. The test will be done for each direction. The boundary condition at two opposite sides are set to different head values, the other both boundary condition to q = 0. A linear head function should be the result. Same effect should have a system with different hydraulic conductivities such as shortly described in test case 3. Setting all boundary conditions by head values with different values, a 1D linear solution should appear in the middle section of the model domain.

Test case 7 checks the impact of the source/sink term. Specified in the middle of the model domain with fixed head at all four boundaries, the shape of the head distribution should be like a circle in the middle changing towards a square shape in the directions to the boundaries. Specified at other locations within the model domain similar head distribution figures influenced by the boundary geometry will be expected. Located directly at a boundary with given head value, the source/sink term should have no impact on the result. Located on a boundary with given flux the system should give a warning, as the boundary conditions for flux and a given source/sink term are using different derivation order of the head variable and different physical units.

The specification of test cases can be continued by more complex combinations of parameters. But this is out of the scope of these lecture notes. The introduced test cases 1–7 illustrate the strategies to test the basic functionality of the software system. For commercial software development an extensive test check case catalogue should be set up. The application of the test cases can be implemented in a test application for an automatic check run of the system. This should be started after each change of the system and might help during the first implementation of the system as well as for later adaptations and extensions of the system.

Overview Test Cases 1-7

TC	N_1	N_2	L_1	L_2	K_{11}	K_{22}	w	BC N	BC S	BC E	BC W	Result
1.1	3	3	1	1	1	1	0	h 0	h 0	h 0	h 0	H=0
1.2	7	7	..	..	..	..	..	..	..	..	..	..
1.3	10	10	..	..	..	..	..	..	..	..	..	..
1.4	100	100	..	..	..	..	..	..	..	..	..	..
1.5	100	205										
1.6	205	100										
2.1	50	50	1	3.5	1	1	0	h 0	h 0	h 0	h 0	h=0
2.2	50	50	3.5	1	..	..	..	..	..	..	..	..
2.2	50	50	3.5	4.1	..	..	..	..	..	..	..	..
2.3	81	81	1	3.5	1	1	0	h 0	h 0	h 0	h 0	h=0
2.4	81	81	3.5	1	..	..	..	..	..	..	..	..
2.5	81	81	3.5	4.1	..	..	..	..	..	..	..	..
3.1	50	50	1	1	1	21	0	h 0	h 0	h 0	h 0	h=0
3.2	50	50	..	..	21	1	..	..	..	..	..	..
3.2	50	50	..	..	21	13	..	..	..	..	..	..
4.1	50	50	1	1	1	1	0	h 2	h 2	h 2	h2	h=2
4.2	50	50	..	..	..	..	..	q 0	h 2	h 2	h2	h=2
4.3	50	50	..	..	..	..	..	h 2	q 0	h 2	h2	h=2
4.4	50	50	..	..	..	..	..	h 2	h 2	q 0	h2	h=2
4.5	50	50	..	..	..	..	..	h 2	h 2	q 0	q0	h=2
4.6	50	50	..	..	..	..	..	q 0	q 0	h 2	h2	h=2
4.7	50	50	..	..	..	..	..	q 0	h 2	q 0	h2	h=2
4.8	50	50	..	..	..	..	..	q 0	q 0	q 0	h2	h=2
4.9	50	50	..	..	..	..	..	q 0	q 0	q 0	q 0	singul.
5.	50	50	..	..	..	..	..	h 1.1	h 1.5	h 1.3	h 1.6	bound.
6.1	50	50	..	..	..	..	..	h 1.1	h 1.5	q 0	q 0	h lin x_2
6.2	50	50	..	..	..	..	..	q 0	q 0	h 1.1	h 1.5	h lin x_1
6.3	50	50	..	..	e-8	1	..	h 1.1	h 1.5	h 1.1	h1.1	h lin x_2
6.4	50	50	..	..	1	e-8	..	h 1.1	h 1.1	h 1.1	h 1.5	h lin x_1
7.1	51	51	..	..	..	..	C 1	h 0	h 0	h 0	h 0	h sym
7.2	51	51	..	..	..	..	1	h 0	h 0	h 0	h 0	h c->r
7.2	51	51	..	..	..	..	N 1	h 0	h 0	h 0	h 0	h =0
7.3	51	51	..	..	..	..	N 1	q 0	h 0	h 0	h 0	error

4. Performance Test

Numerical methods implemented in hydroinformatics systems require computer performance in respect to memory and number crunching. Both can be estimated based on the mathematical description of the applied numerical method and measured during application. The measured data for memory and calculation time depends hardly on the used hardware platform and software/operation system environment. Usually the performance test is done on a “typical” platform to check the expected relationships of required computer memory and performance to the main system properties.

There are several tools available to support the performance test of system applications. Besides internal functions of system development toolkits, external tools provide system performance monitoring, analysis and reporting functionalities. Typical examples are profiler or debugger. These lecture notes did not consider these tools for commercial software development. Some simple “manual” performance test methods for Java applications will be used as examples for academic/research oriented system development.

4.1 Example FDM 2D

The performance properties of the introduced finite difference method example for 2D groundwater flow are mainly specified by the classes/objects to describe the FDM 2D model, the method to set-up and solve the equation system and the equation system itself. The impact of the different model parameter can be analysed and described by related relationships. These relationships can be tested by doing related measurements with the developed software tool.

Memory

Memory measurement in Java can be done simplified by a method of the Runtime class:

```
Runtime.getRuntime().totalMemory()
```

As the internal memory management in the Java environment is implemented in the garbage collector, the Runtime method returns the actual managed memory. This might be different from the actual required memory of the hydroinformatics system. Typical behaviour of garbage collectors is to require new memory in larger blocks. However, the result of the Runtime method can be used to get the dimension of the required memory during the different steps of the system performance.

The total amount of memory can be estimated by the used objects and their variables. Critical for memory are arrays and sets of objects with larger numbers of related entities. The number of nodes is the critical attribute in case of FDM 2D. The number of node objects with coordinates and physical state variable h_n, as well as the number of source/sink term is proportional to the node number. The node number N in 2D is related product of the two node numbers in each direction N_1 * N_2. Assuming a similar node number in both directions the total node number of the system is proportional to the number of nodes in one direction N_{1D}.

$$N \sim N_{1D}^2$$

The equation system consists of two vectors and one matrix. The size of both vectors is proportional to the node number N. The dimension of the matrix is N which leads in case of the full storage of the matrix to N^2.

(8-1) $$mem\ \boldsymbol{K} \sim N^2 \sim N_{1D}^4$$

Using 8 bytes per element, typical numbers of memory for the sparse matrix are:

N_{1D}	N	Memory Sparse Matrix MB	Memory Band Matrix MB
10	100	0.1	
20	400	1.2	0.1
50	2500	48.0	1.0
100	10000	762.5	7,6
200	40000	12208.0	61,0
500	250000		954.0
1000	1000000		7630.0

As most matrix elements are 0, other storage structures might be useful. Example is a band matrix with a specified number of diagonals (bandwidth b). For FDM groundwater 2D the bandwidth is specified by the number of nodes in direction 1. Using this approach the memory size of the matrix is defined by:

(8-2) $$mem\ \boldsymbol{K} \sim b\ N \sim b\ N_{1D}^2 \sim N_{1D}^3$$

Changing the approximation level by a factor of 2 for the number of nodes in both directions, the number of nodes will increase by a factor of 4 and the required memory space for the matrix will increase by a factor of 8.

Calculation Time

Time measurement in Java can be done simplified by a method of the System class:

```
System.currentTimeMillis()
```

The method returns the actual system time in milli seconds. Using this before and after critical code parts, the difference is the used time. However, this time is the total time and might consider parallel garbage collector operations, operation system activities and any other process performance on the computer.

The required calculation time is mainly specified by the set-up of the equation system and solving the equation system. Setting-up the equation system is a loop on all nodes with the related finite difference approximation equation. This is proportional to the node number N.

(8-3) $time\ \boldsymbol{ES}_1 \sim N$

$\boldsymbol{ES_1}$ Method to set up the linear equation system

Solving the equation system requires a decomposition of the matrix and forward-backward elimintaion of the unknown variables. For a sparse matrix the calculation time is proportional to N to the power of 3; the forward-backward elimination to N to the power of 2.

(8-4) $time\ \boldsymbol{ES}_2 \sim N^3$

$\boldsymbol{ES_2}$ Method decomposition sparse matrix

(8-5) $time\ \boldsymbol{ES}_3 \sim N^2$

$\boldsymbol{ES_3}$ Method forward-backward elimination

Equations (8-3) to (8-5) show, that the decomposition effort is critical for the calculation time for this implicit numerical method.

(8-6) $time\ \boldsymbol{FDM2D} \sim N^3 \sim N_{1D}^6$

$\boldsymbol{FDM2D}$ finite difference method for 2D groundwater flow

Changing the approximation level by a factor of 2 for the number of nodes in both directions, the calculation time for the whole system will increase by a factor of 64 !

As described in the analysis of the memory the performance can be enhanced by using a bandmatrix structure.

(8-7) $time\ \boldsymbol{ES}_2 \sim b^2\ N$

(8-8) $time\ \boldsymbol{ES}_3 \sim b\ N$

The decomposition effort is still critical for the calculation time for this implicit numerical method. However the exponent has changed significant:

(8-9) $time\ \boldsymbol{FDM2D} \sim b^2\ N \sim N_{1D}^4$

Changing the approximation level by a factor of 2 for the number of nodes in both directions, the calculation time for the whole system will increase by a factor of 16.