



# An integrated technology of automated verification and testing

**Motorola**

**Pavel Drobintsev**

**(Pavel.Drobintsev@motorola.com)**

# Outline

**The problem**

**The common approach to software development process**

**Brief technology description**

**Step by step technology usage**

**Conclusions**

# The problem is:

**Manual or semi automated testing is a bottleneck for software development process**

- Testing takes up to 40% of project time**
- Manual testing strongly depends on a human!**
- Regression testing is a big problem for modern devices**

**Cost of defects correction at later phases is higher than at the earlier**

- Cost of defect correction grows exponentially**

# Technology goals

**Guaranteeing software quality via integration of verification and testing automation.**

## **Verification:**

Finding Incorrectness (inconsistencies – absence of non-determinism, incompleteness – absence of deadlocks) in Requirements and Specifications.

Localization and correction of findings on the earlier phases before start of the coding.

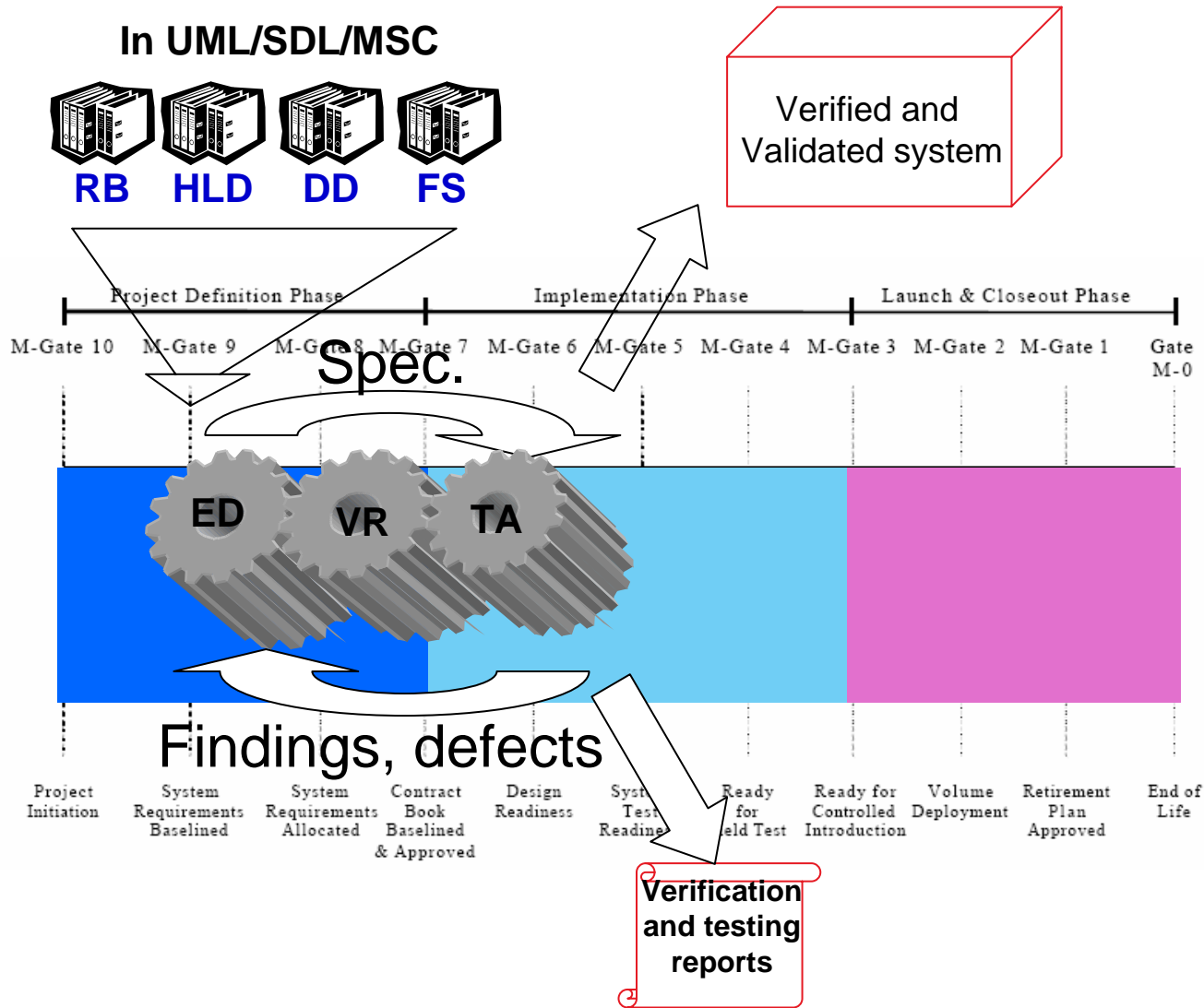
Optimal Formal Tests Generation with 100% coverage of needed features.

## **Testing Automation:**

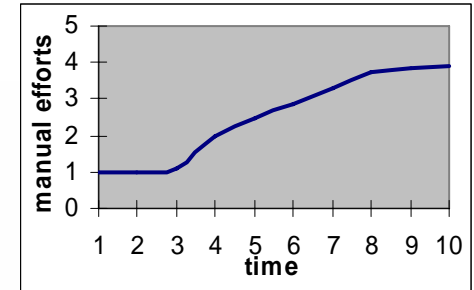
Target tests code generation.

Fully automated test cycle.

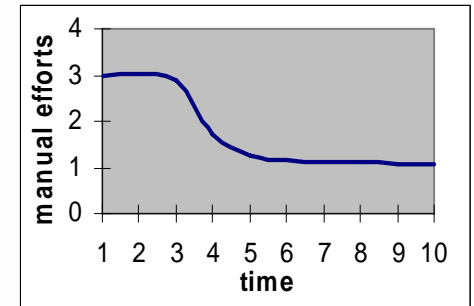
# Technology usage area



## Manual efforts



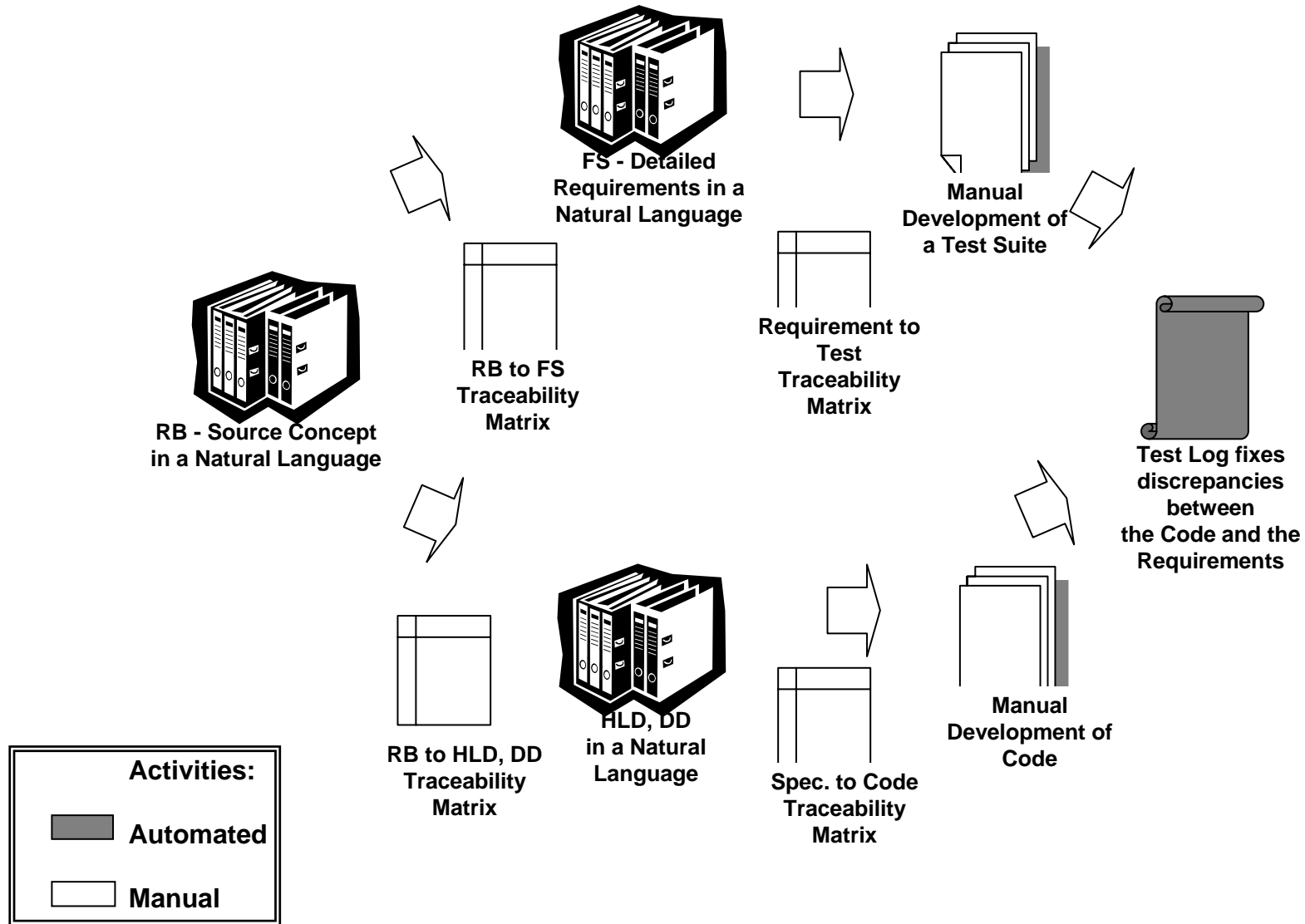
## without Technology



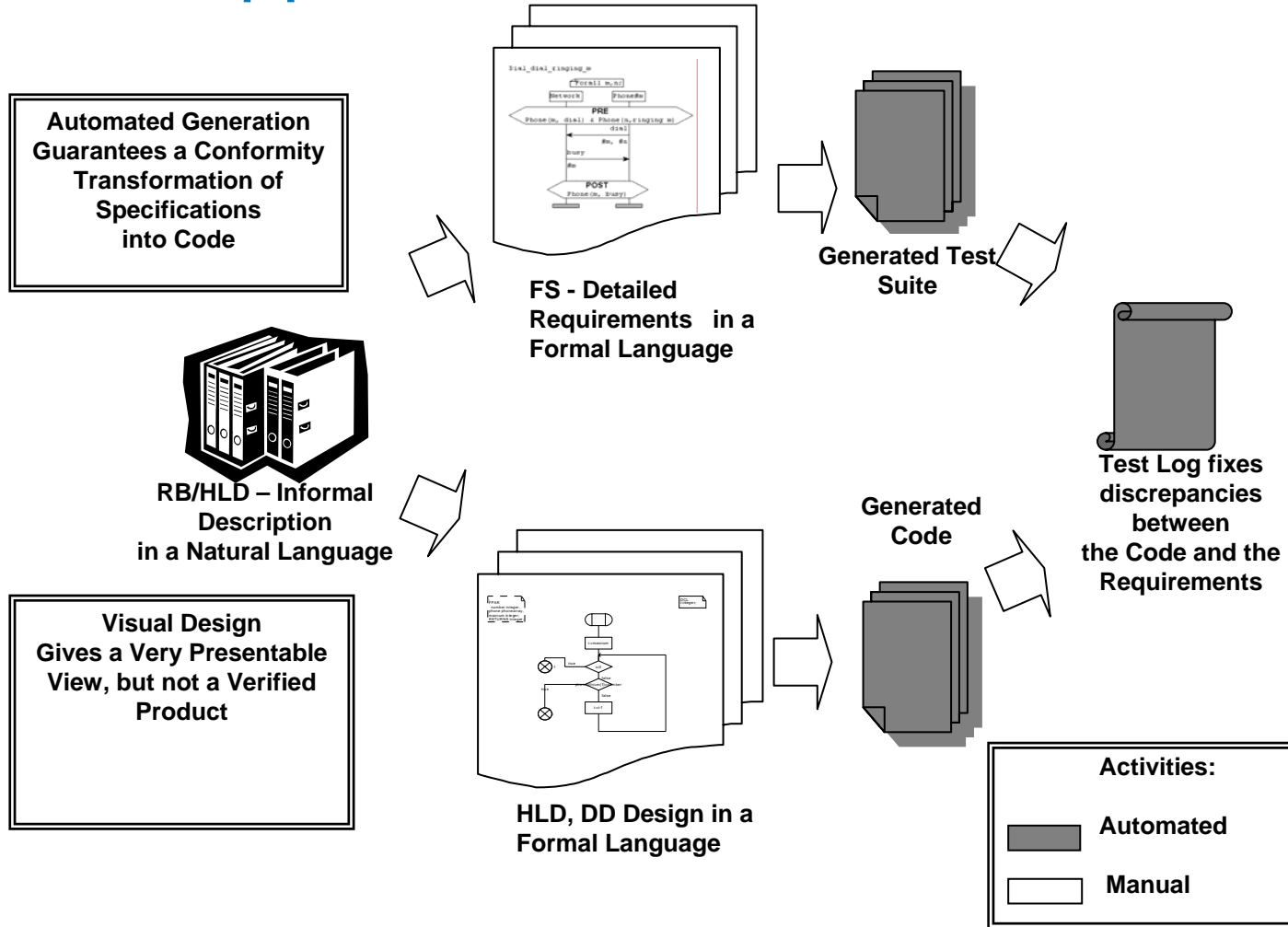
## with Technology

**Major manual efforts shift to earlier phases**

# Conventional approach to software development

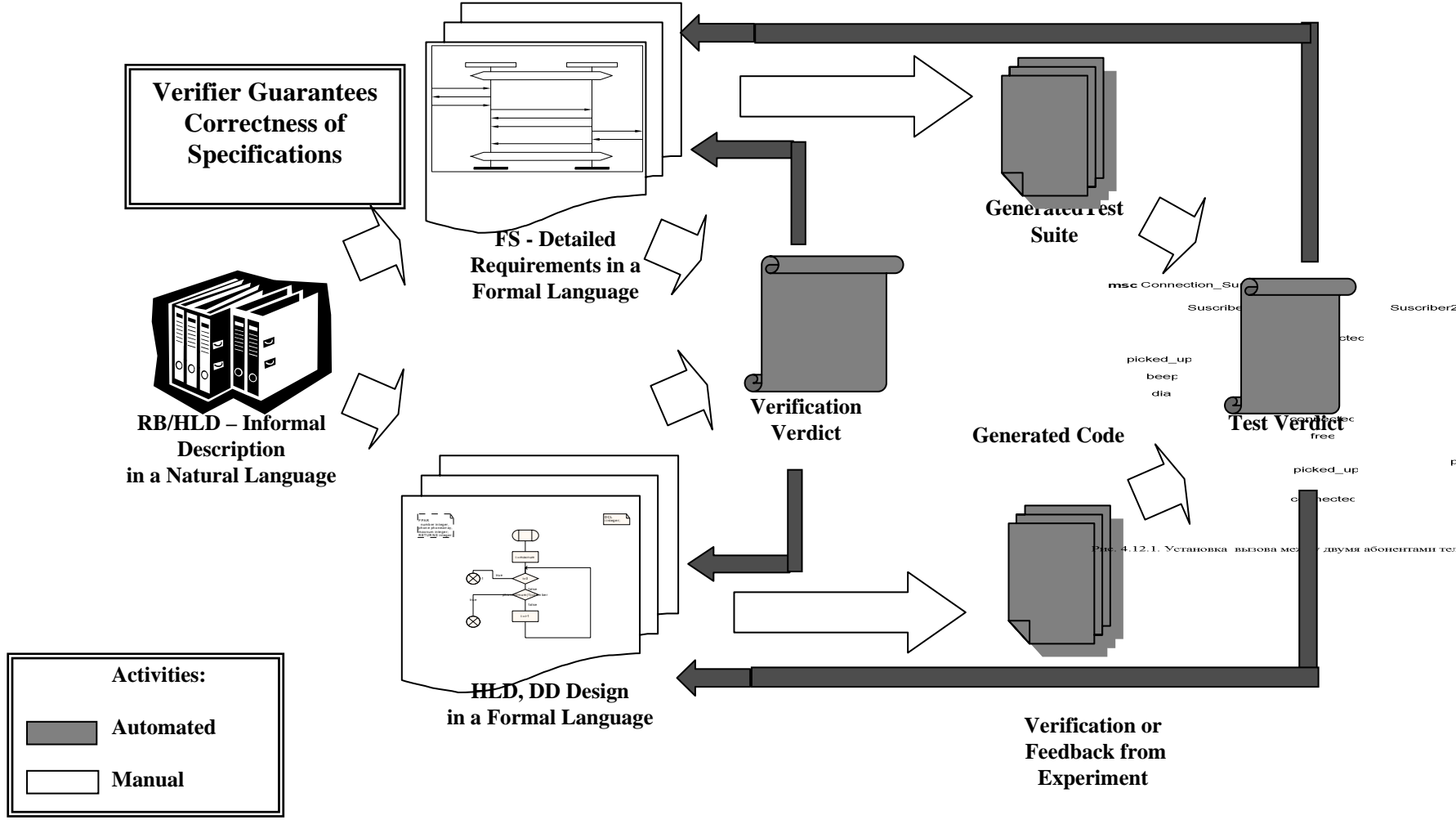


# Suggested approach



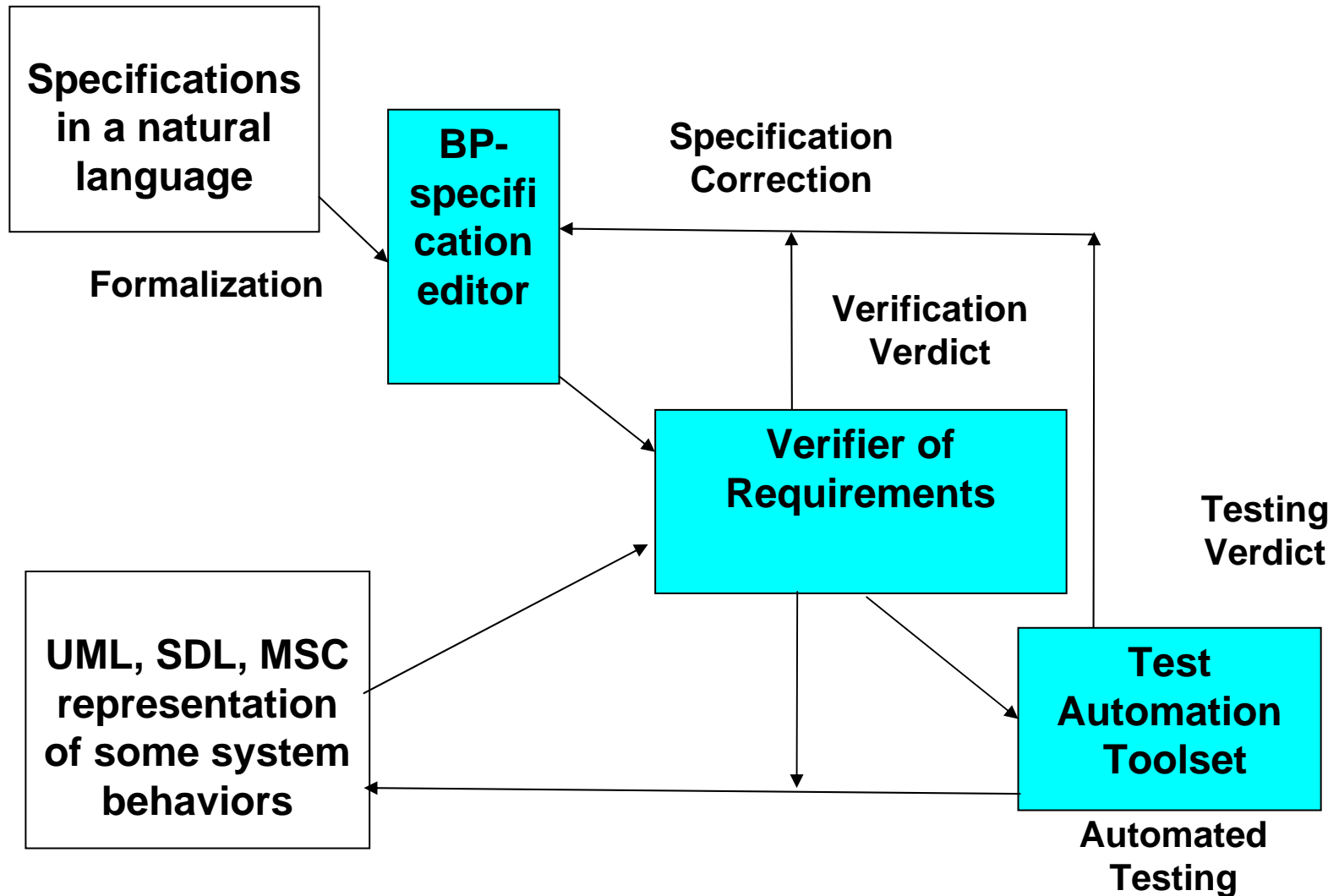
## Automated code and test generation

# The technology chain



**Verification and testing complement each other**

# Step by step Technology description



# Step by step technology description

## step 1

### **Step 1 – Formalization of requirements**

**Input:** Set of requirements in a natural language.

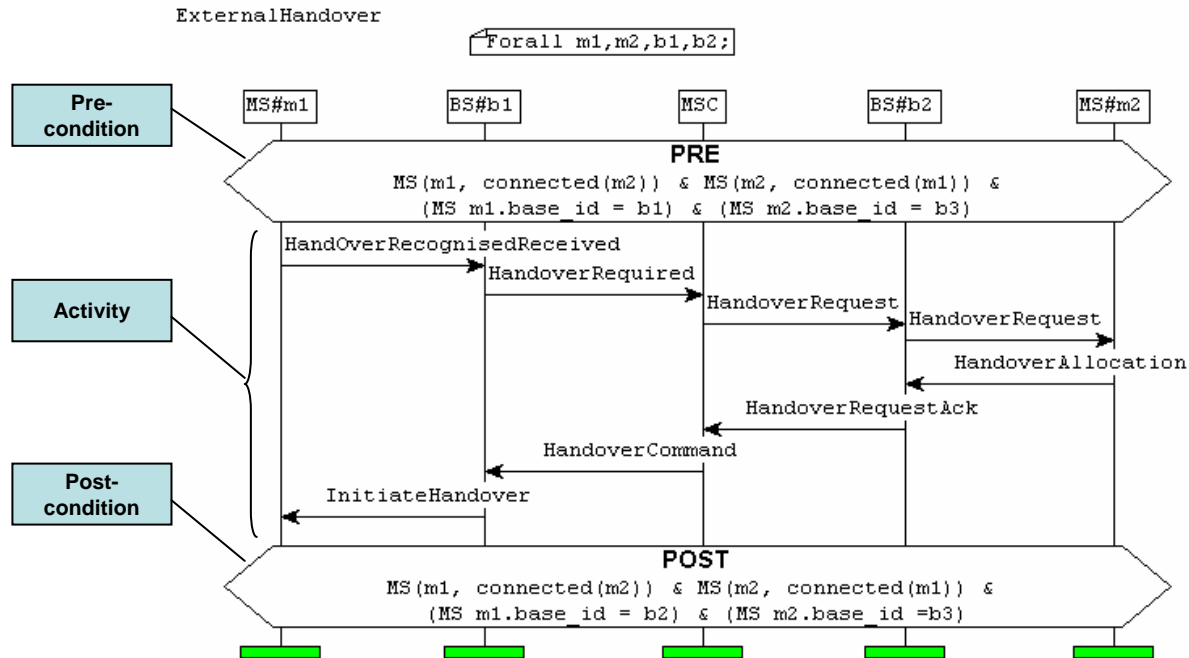
**Output:** Set of basic protocols or SDL, MSC, UML diagrams.

**Added value:** Set of manually found discrepancies and inconsistencies.

### **Description:**

The behavioral requirements created in a natural language are translated into the formal languages MSC, SDL and UML, widely used for system behavior protocols description.

# Basic protocols



## Requirements transformation into Basic Protocols

# Step by step technology description

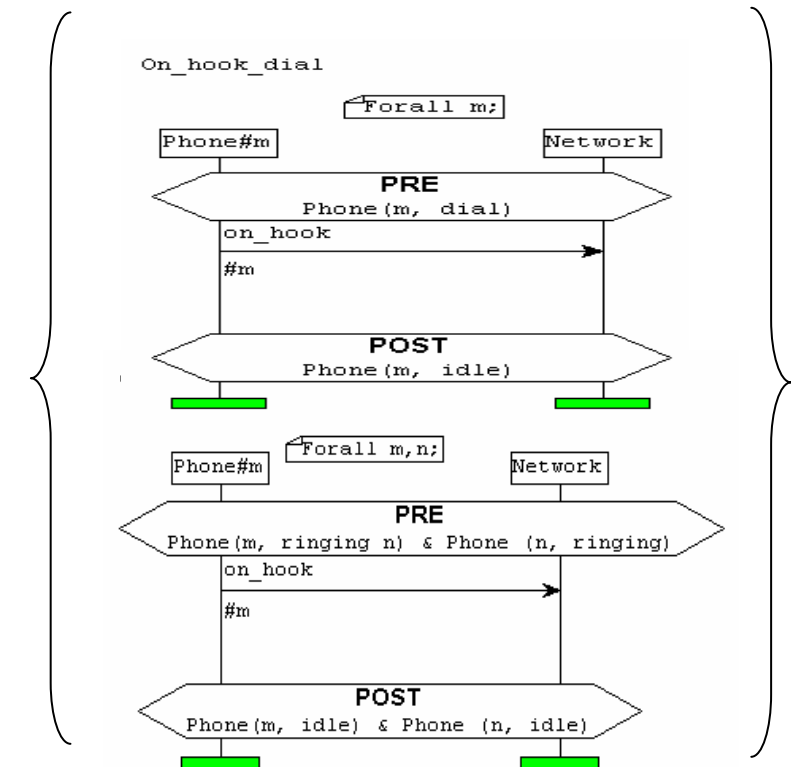
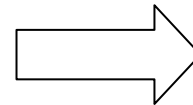
## step 1

Requirements + formalization methods  $\Rightarrow$  basic protocols

**R1. Dialing phone became idle after putting the receiver on hook.**

**R3. If a telephone m is in the ringing n state and puts receiver on hook both telephones will turn into the idle state.**

Formalization process



Informal languages and procedures

Formal languages and procedures



# Step by step technology description

## step 2

### **Step 2 – Basic protocols verification**

**Input:** Set of basic protocols.

**Output:** Verdict on the protocols' consistency.

**Added value:** (1) automatically found discrepancies and inconsistencies; (2) a set of consistent basic protocols.

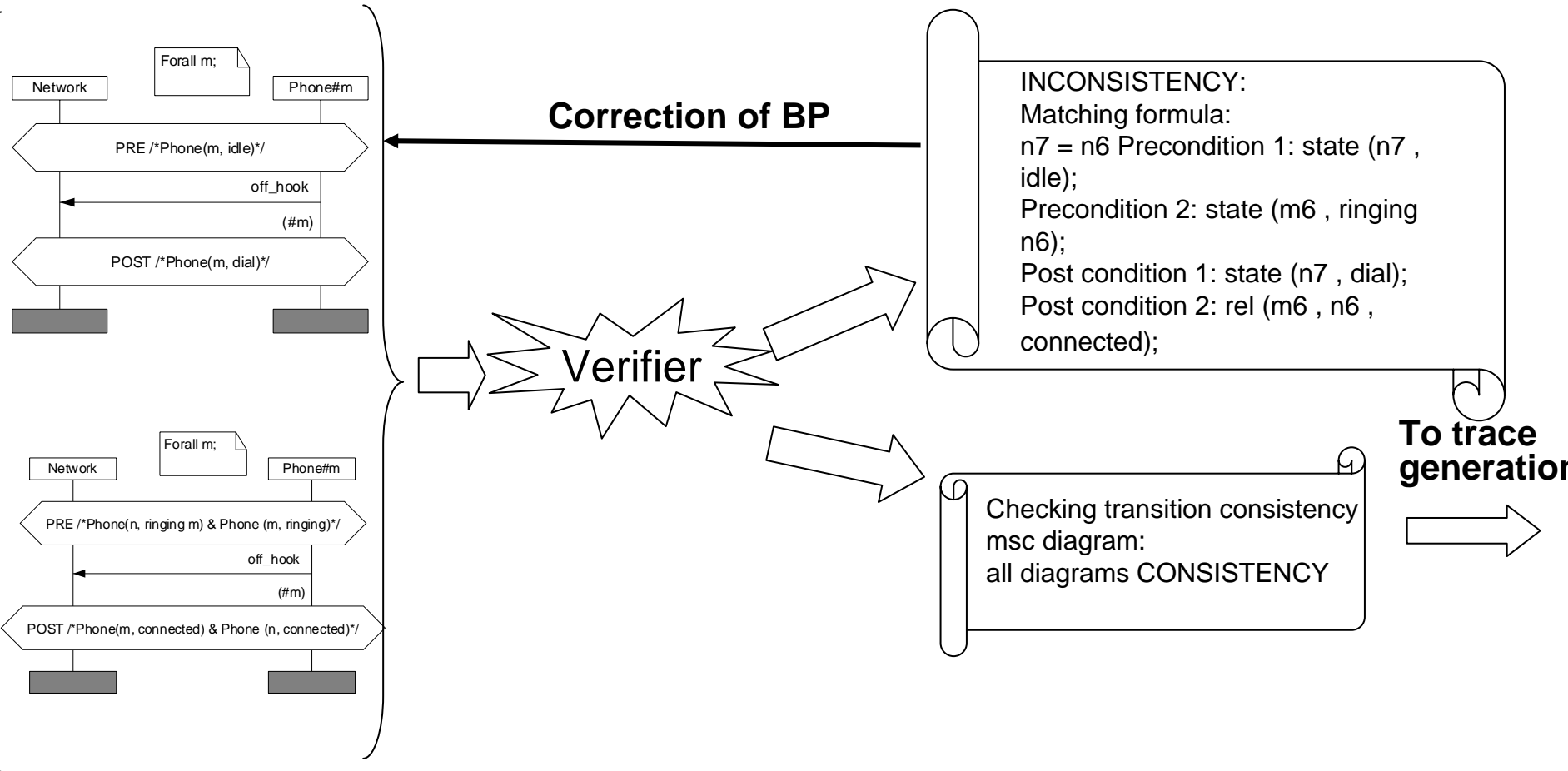
### **Description:**

The created basic protocols are checked by the Verifier.

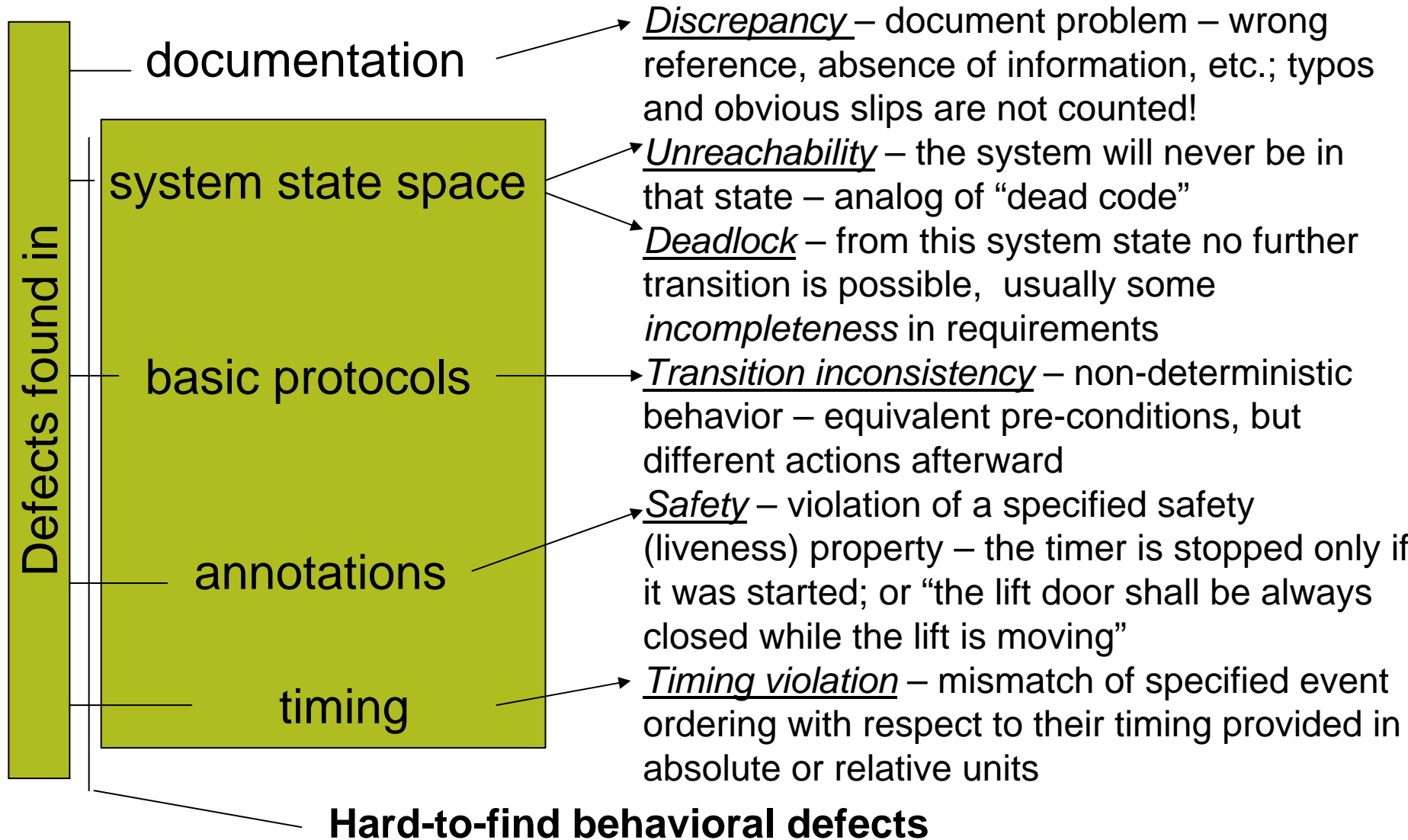
# Step by step technology description

## step2

Set of basic protocols + Verifier features  $\longrightarrow$  List of inconsistencies



# Types of defect found with the technology



# Step by step technology description

## step 3

### Step 3 – Traces generation

**Input:** Set of verified and consistent basic protocols;

**Output:** Set of traces for full coverage of requirements.

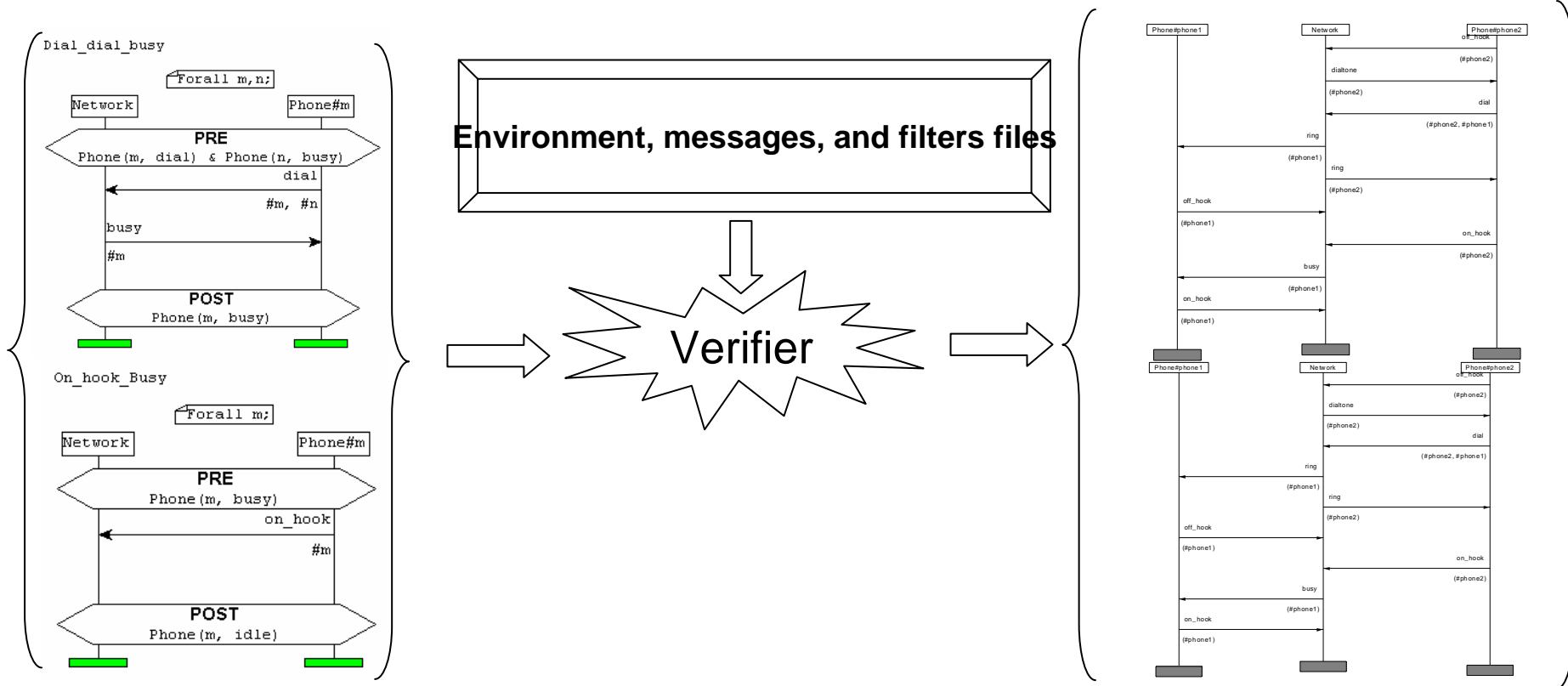
**Added value:** Generated traces which formally cover 100% of the considered specifications and can be used for further generating a respective test suite.

### Description:

To set up the trace generation process, three files should be created:

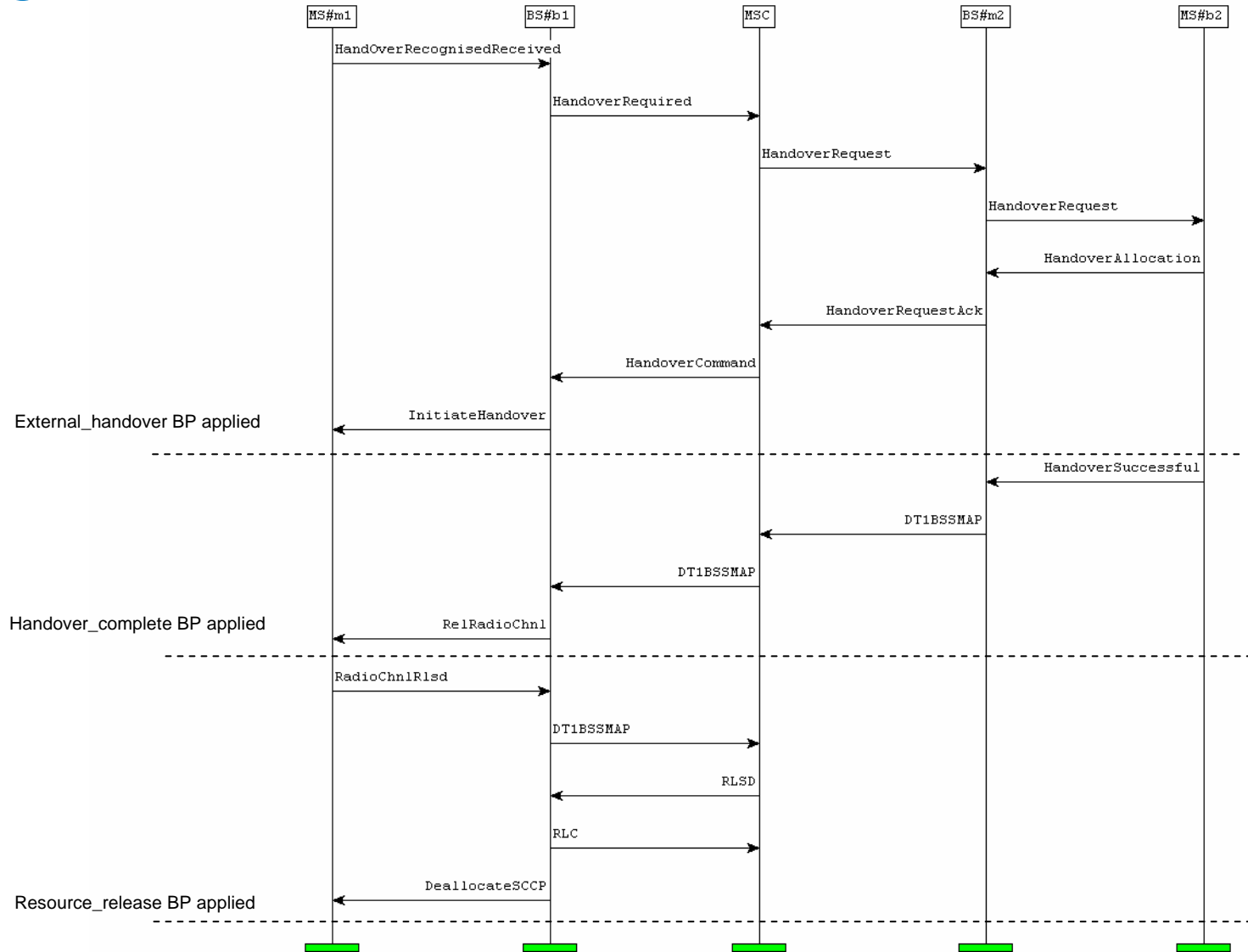
# Step by step technology description step 3

Basic protocols + Configuration files + Verifier features → Set of traces



# Traces

Trace



# Step by step technology description

## step 4

### **Step 4 – Test generation**

**Input:** Set of traces.

**Output:** Set of tests for a full coverage of the specifications.

**Added value:** The automatically generated tests in the target language.

### **Description:**

In the process of test generation the following tools are used:

Abstract test generation tool – which generates tests from MSC traces in an abstract form in the tcl language.

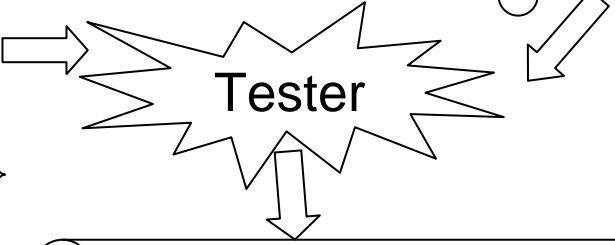
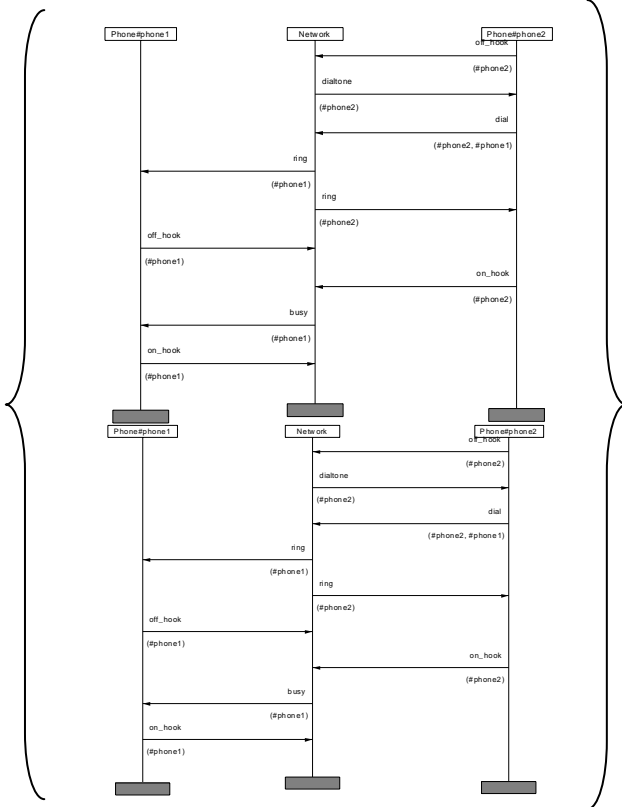
Code generation template – which translates tests from tcl into the selected target language.

# Step by step technology description

## step 4

### Set of traces + Macro definitions + TA features

### Set of tests in "C"



```
<MACRO datatype="number" name="#m"
priority="100" replace="replace" type="list" vector="" >
<VALUELIST condition="" >
<VALUE value="1" />
</VALUELIST>
</MACRO>
```

```
static TAT_TEST_RESULT tat_verdict0004_state0(TAT_TEST_INSTANCE *id)
{
TAT_EVENT_INSTANCE __ev;
memcpy(&__ev, &tat_phonetest_verdict0004_0_event, sizeof(__ev));
if (TAT_Send_OFF_HOOK(id, &__ev, &tat_phonetest_verdict0004_0_event, 2)
!=TAT_TEST_CONTINUE) return TAT_TEST_CRITICAL;
id->nextState=1;
return TAT_TEST_FINISHED;
}
```

# Step by step technology description

## step 5

### **Step 5 – Creation of SDL/UML model**

**Input:** requirements in a formal or informal language.

**Output:** SDL/UML code of a model.

**Added value:** (1) The model can be automatically translated into C code; (2) The SDL/UML code of the model can be widely reused in other projects within the same subject domain.

### **Description:**

Some type of editor is used for working with SDL/UML, Simulator is used for model debugging, and target oriented compiler is used for C code generation.

# Step by step technology description

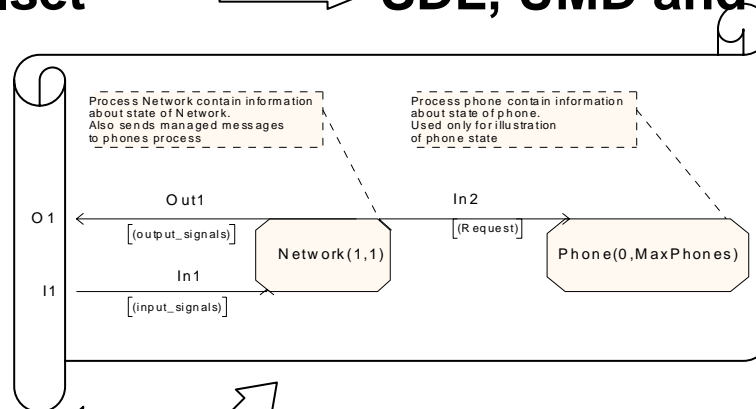
## step 5

Requirements + Editor toolset

→ SDL, UMD and C code of model

**R1. Busy phone became idle after putting the receiver on hook.**

**R3. If a telephone is in the ringing state and puts receiver on hook both telephones will turn into the idle state. (Rn 3)**



**SDL/UML editor**

**compiler**

```
*****
* PROCESS Network
* <<SYSTEM Pots/BLOCK POTS>>
* *****/
#ifdef XCOVERAGE
long int yPrsC_z02_Network[ySym_z02_Network+1];
#endif
XCONST XSIGTYPE yPrsS_z02_Network[] =
{SIGNAL_NAME(on_hook, &ySigR_z4_on_hook),
 SIGNAL_NAME(off_hook, &ySigR_z5_off_hook),
 SIGNAL_NAME(dial, &ySigR_z7_dial);
```

# Step by step technology description

## step 6

### Step 6 – Wrapper generation

**Input:** Set of configuration files.

**Output:** Wrapper for the SDL model in C.

**Added value:** The wrapper connects the test and the model interfaces.

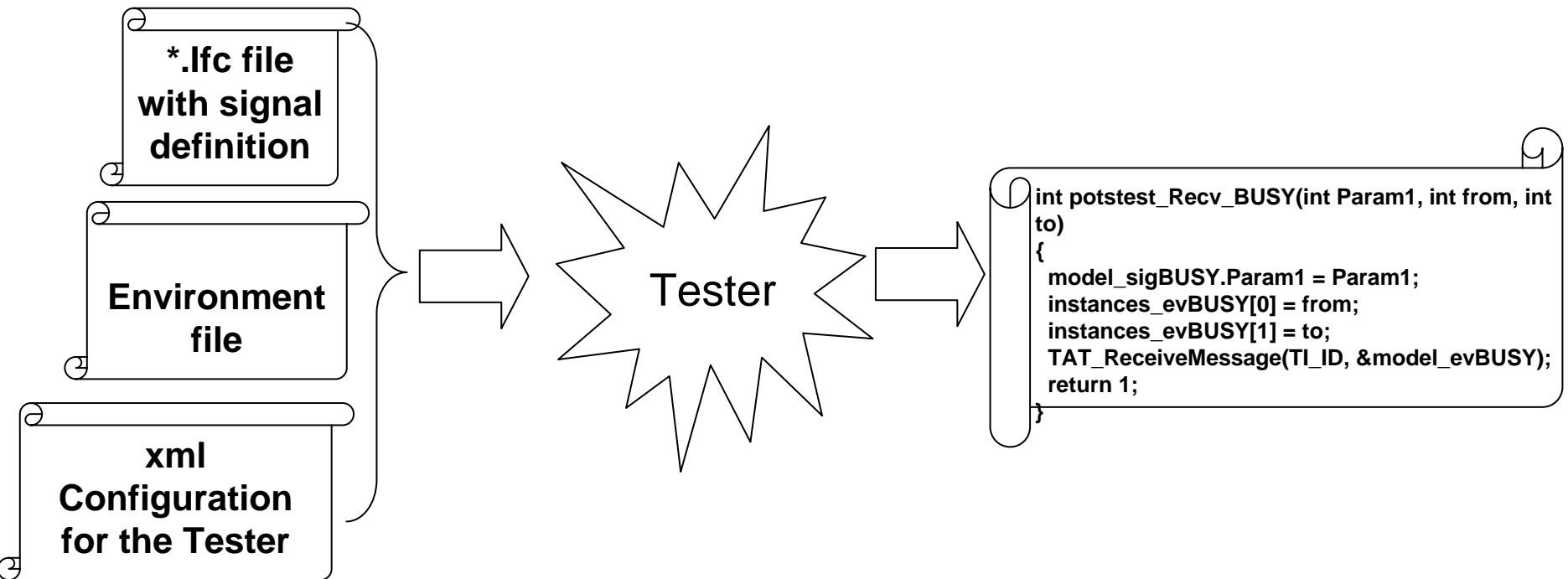
### Description:

A wrapper for an SDL\UML application is generated automatically and is compiled in one unit with the C code of the application under testing. All message processing is implemented in the code of the wrapper.

# Step by step technology description

## step 6

Configuration files + Tester features  $\longrightarrow$  SDL- MSC wrapper



# Step by step technology description

## step 7

### Step 7 – Test suite execution

**Input:** Set of generated tests in the target code, code of the wrapper, and C code of the SDL system.

**Output:** Test suite execution verdict with found errors

**Added value:** The output graphical verdict is compatible with the verifier input and can be resubmitted for further analysis.

### Description:

#### Two types of verdicts are returned:

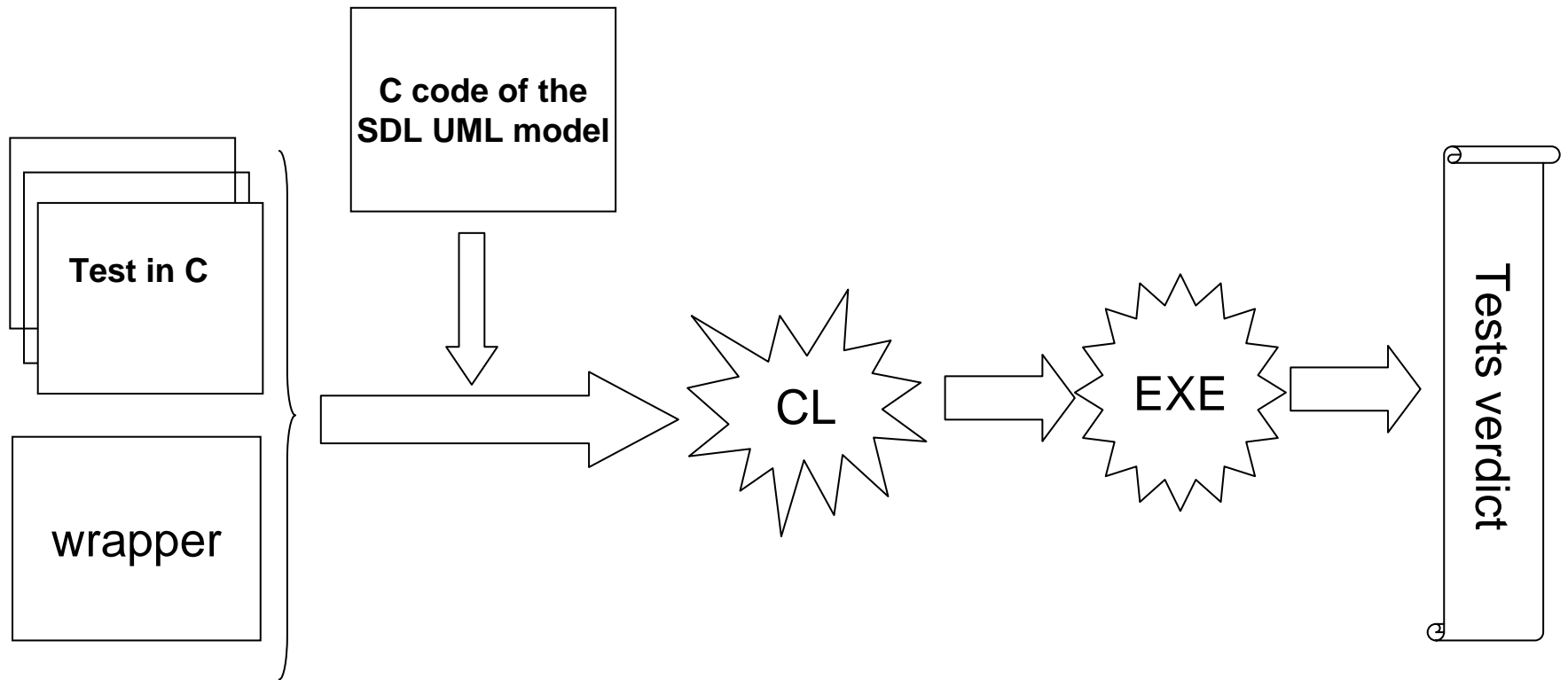
Textual verdict – contains messages set to and received from the application with identification of the error (if it exists)

Graphical verdict – same found errors in form of MSC traces leading to the point of error.

# Step by step technology description

## step 7

Tests + Model code + Wrapper+ C compiler  $\longrightarrow$  Tests verdict

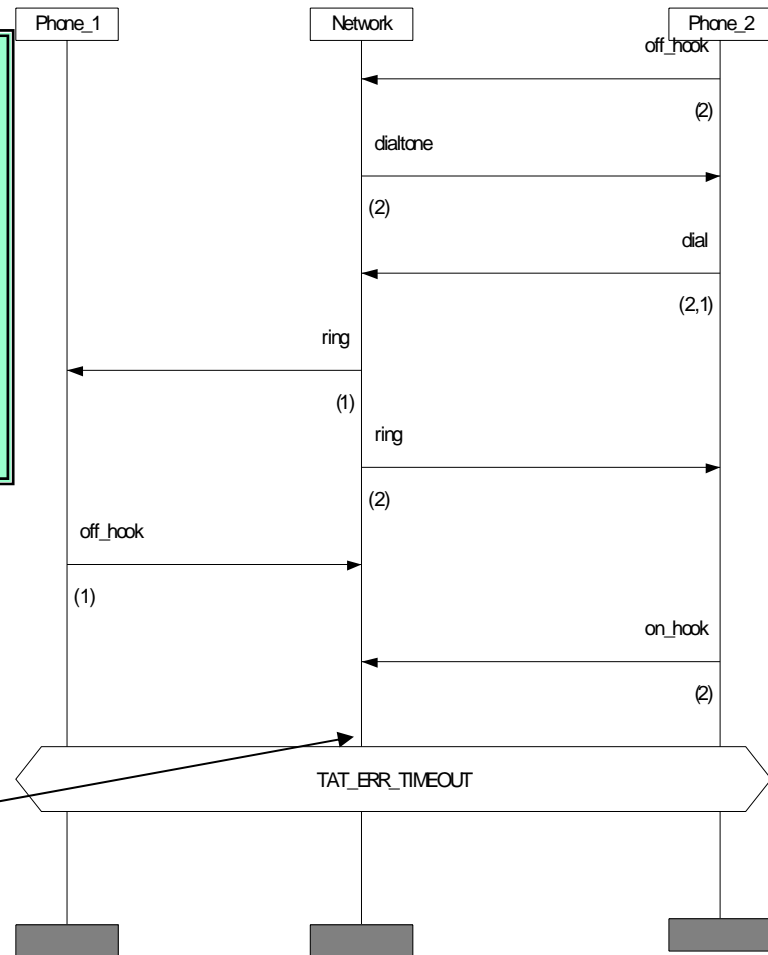


# Textual and graphical test verdicts

```
RMRK: Test case verdict0004 started. (Iteration:0)
{000000}[0007] SEND: OFF_HOOK(2) <PHONE_2->NETWORK>
{000000}[0009] RECV: DIALTONE(2) <NETWORK->PHONE_2>
{000000}[0011] SEND: DIAL(2,1) <PHONE_2->NETWORK>
{000000}[0015] RECV: RING(2) <NETWORK->PHONE_2>
{000000}[0013] RECV: RING(1) <NETWORK->PHONE_1>
{000000}[0017] SEND: OFF_HOOK(1) <PHONE_1->NETWORK>
{000000}[0019] SEND: ON_HOOK(2) <PHONE_2->NETWORK>
ERROR: Timeout error
RMRK: Test 1 finished. Status=TAT_ERR_TIMEOUT (Iteration: 0)
(Iteration: 0)
```

System does not send a busy signal to phone 1

**Test FAILED**



# Conclusions

**Automated generation of test suites and their run allow to save at least 50% of the testing phase time.**

**The technology provides a complete test coverage of the behavioral properties of the system under testing through automatically generated traces.**

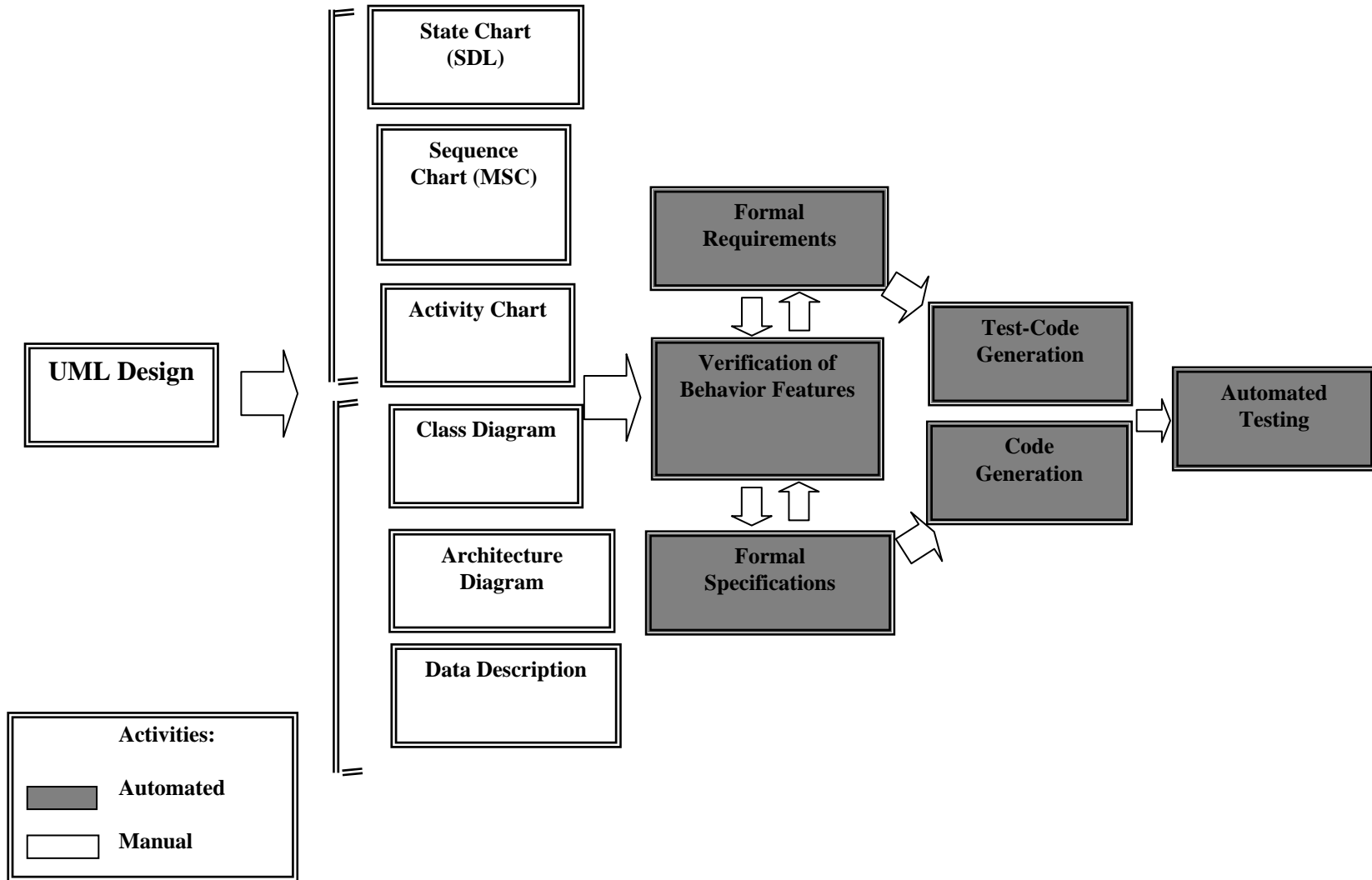
**A new option of extending the system functionality in high-level languages for better specification understanding was implemented in the scope of the technology.**

**Security auditing with technology becomes possible.**

# Thank you

# Backup slides

# UML technology mapping



## Formal Description from Informal Documentation