# Black Box Test Report

## 1 Telephone Switches of Group Denver

### 1.1 Verifying the requirements

After we received the executable, the configuration file and the SRS of group Denver we started with verifying the SRS. The group did well on formatting and the detailed list of the software requirements. However, we also found quite some ways in which the SRS could be improved. First of all, we would like to say something about the grammar. We found quite some mistakes, like "The user can specify his initial setup via file". This led to a discussion in what was meant here. Did they mean to say "a file"? Or did they forget the word that specifies the file, because they meant the configuration file?

Secondly, it was not always clear what was meant with a certain word. For example, in the sentence "The product is independent and totally self-contained." Since the word "product" was not further defined, we were not sure what was meant by this word. It could refer to the executable or as the executable together with the configuration file and the SRS.

Next to that they were not always precise, when looking the sentence "The initial setup consist of a non empty set of nodes, an arbitrary number of phones." they didn't specify a maximum for the number of phones nor of the number of nodes. This is quite dangerous, since it is most likely that there is a maximum.

It also looked like the words "system" and "handler" were mixed up. In the purpose they said: "This project simulates a system responsible for routing telephone calls." After that, the handler was specified by the following: "The handler is the central point where all phones and nodes are registered and it is responsible for the routing of calls" But later on we read the following: "A malfunctioning phone can not be contacted at all by the system," Since the handler routes the calls, we would think the handler contacts the phones.

Lastly, the group referred to phones as XX though the phone IDs in the executable consist of four numbers.

### 1.2 Black box unit testing

During black box testing the program is tested if the program has the requirements stated in the SRS. During the testing we noticed that the program does not behave the same way each time when the same arguments are supplied. This shows that the program is not deterministic and that there is some kind of randomness involved in the program. For testing this makes things harder since the result of a certain test may differ each time the test is ran. To replicate the results in this section, note that one might be required to re-run the test several times to find the same results as stated.

### 1.2.1 Configuration file testing

The configuration file contains the parameters for the program when it gets executed. The program expects the file to be valid JSON, but the file may contain raw input, so this is a vulnerable point of the program. All of the following tests passed successfully:

- trying to execute the program without a file yields correct error,
- non-JSON file yields correct error,
- valid JSON file containing no phones or nodes parameter yields correct error,
- valid JSON with 10 phones and 10 nodes is correctly loaded,
- valid JSON with 10 phones and less than 1 node yields correct error,
- valid JSON with less than 1 phone and 10 nodes yields correct error,
- valid JSON with a string as input for phones and 10 nodes yields correct error,
- valid JSON with 10 phones and a string as input for nodes yields correct error,
- valid JSON with 10 phones, 10 nodes and other non-program related variables is correctly loaded,
- a file with an attempt to inject code does not execute the code, which is correct.

### 1.2.2 Program behaviour testing

Once started with a valid configuration the program accepts commands from the user. All available commands are given in the documentation and are call, answer, hangup, phone_offline, phone_malfunction, phone_online, node_malfunction and node_online. All of the following passed successfully with a valid configuration file:

- when an user submits a random string as command then the program yields a correct error,
- when an user submits a command with an invalid parameter, then the program yields a correct error,
- when an user submits a command with too few parameters, then the program yields a correct error,
- when an user submits a command with too many parameters, then the program yields a correct error,

- when an user submits a command with correct parameters, it will work as expected[1].
- when an user submits a command which should not run[2], then the program yields a correct error.

## 1.3 Found unwanted behaviour

We have ordered the found unwanted behaviour in order of severity.

### 1.3.1 Low severity

The bugs in this category are mostly incorrect feedback or other unexpected behaviour which do not influence the program in a negative way.

The command "call not_a_phone not_a_phone" yields error "Phone cannot call itself", but are not phones.

When a function is called without arguments, but multiple spaces we get an error which says that too many arguments have been submitted.

A command starting with a space does not get executed.

### 1.3.2 Medium severity

This category contains the bugs which negatively influences the functionality or limits the program in an unneeded manner.

We cannot have more than 9999 phones, but this is not specified.

We cannot have more than 9999 nodes, but this is not specified.

### 1.3.3 High severity

This category contains the bugs which are not acceptable, i.e. the program gives wrong output, crashes or has other unexpected behaviour.

When the program is initialized with ten phones and a single node, then sometimes some phones are not able to call each other. Whereas this is already unexpected behaviour, there is more to it. For example the situation may occur that phone 0001 can call both phones 0008 and 0009, which shows that 0001 can connect to both 0008 and 0009 over some node. One might expect that phone 0008 and phone 0009 can call each other over the same node, but this

[1] For exceptions, see the next sub-section on Found unwanted behaviour.

[2] For example, command phone 0000 to call phone 0001 two times concurrently, which should not be possible.

shows impossible.

For this bug we initialize the program with two phones and two nodes. We command phone 0000 to call phone 0001 and get the feedback that the call goes over Node_1. Now we add a malfunction to Node_1 so that it should not be able to use Node_1 anymore. Since Node_1 is malfunctioning, any new call from phone 0000 to phone 0001 should find a new route and hence go over Node_0. However, when we command phone 0000 to call phone 0001 we still get the feedback that the call goes over Node_1.

Again for this bug we initialize the program with two phones and two nodes. Again we command phone 0000 to call phone 0001, but now we command phone 0000 to malfunction, which actually malfunctions phone 0001, and then command phone 0001 to go offline. The program will give the feedback that phone 0001 stopped ringing and that phone 0000 is still dialing phone 0001. One would however expect that also phone 0000 stops dialing. After this one can crash the program with the command phone_malfunction 0000.