1 | INDUSTRIAL CASE STUDY

To compare the trade-off between the required effort and performance of the utilized text mining and clustering techniques, we designed an industrial case study at Alstom Transport AB in Sweden, by following the proposed guidelines of Runeson and Höst[1]. Alstom provides various levels of testing in both manual and automated approaches, where integration testing is performed completely manually. Test cases are created by test engineers following the requirement specification. The number of required test cases for testing a product at BT is rather large and the testing process is performed in several testing cycles. In this study, we analyze all designed integration test cases for testing an ongoing testing project at Alstom called $BR490$ project. As highlighted before, integration test cases are usually functionally dependent on each other, and knowing the dependencies and similarities between them can help us to optimize the testing process. The functional dependencies between the integration test cases for the $BR490$ project are previously detected by us by analyzing the internal signals communications between the software modules (see[2]).

1.1 | The Ground Truth

As stated earlier, the dependencies between requirements and test cases are detected by analyzing the signal communications between software modules[2]. In this regard, the structure of the Train Control Management System (TCMS) which is the testing platform at Alstom is analyzed by us. Figure 1 depicts a part of the TCMS platform, where the traceability graph between software modules, requirements, and test cases is visible.

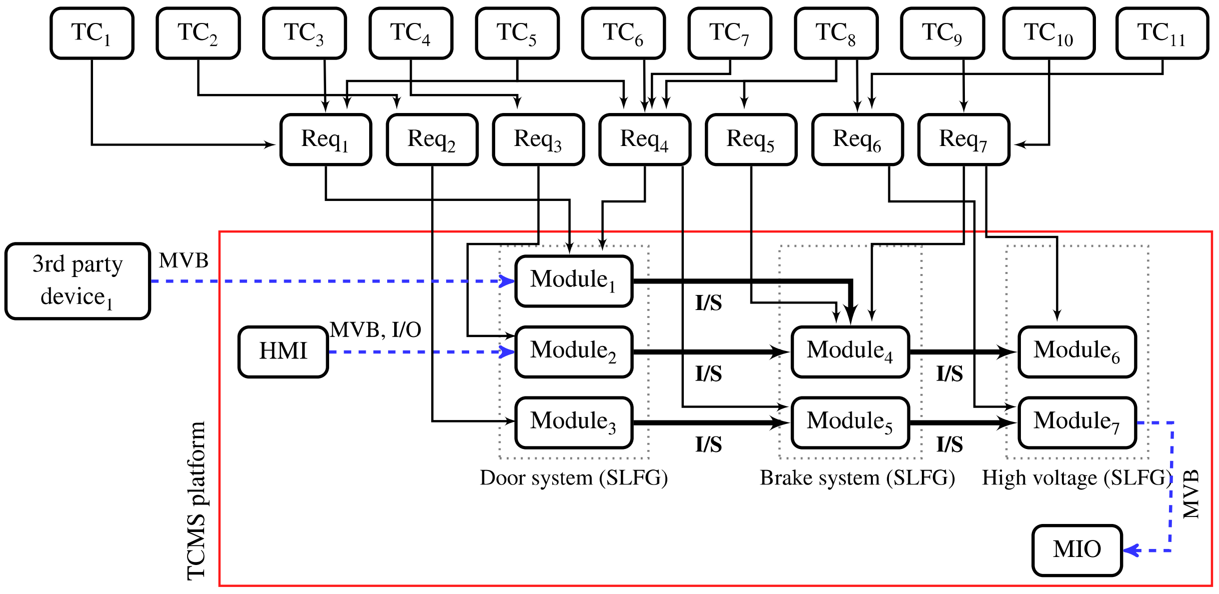

FIGURE 1 The input-output signals, requirements, and test cases in the testing platform (TCMS).

According to our definition, two test cases are functionally dependent on each other if the output *internal signal* from the corresponding software module enters as an input *internal signal* to another corresponding software module. Indeed, if there exists any shared *internal signal* between two software modules (transmitting and receiving the same *internal signal*), then these two modules are depending on each other. Thereby, there is a dependency between all requirements and test cases that are assigned to test those software modules. For instance, TC_7 is functionally dependent on TC_2 because an internal signal from $\textbf{Module}_3$ enters to $\textbf{Module}_5$. The proposed approach in[2] was applied on the $BR490$ project and the results are utilized in this paper as the ground truth, where we know the dependencies between test cases.

Generally, the functional dependency occurs between integration test cases and has a direct effect on the test execution results. Given that test case, TC_2 is functionally dependent on test case TC_1, if TC_1 fails during the testing process, TC_2 will fail as well. This kind of dependency has been observed in several cases in different domains[3,4]. Therefore, detecting the dependencies between test cases is important, since it can be utilized later for ranking them for execution based on their relationship with other test cases. Generally in this approach, test cases are connected and following each other in succession as a directed graph (chain). Thus, independent test cases (e.g. TC_1 in Figure 2) should be executed first and then all other dependent test cases. To clarify the explanation, let us consider a dummy example[1], with five test cases: TC_1, TC_2, TC_3, TC_4, and TC_5. Let us assume that we can describe the following embedded directed graph of dependencies:

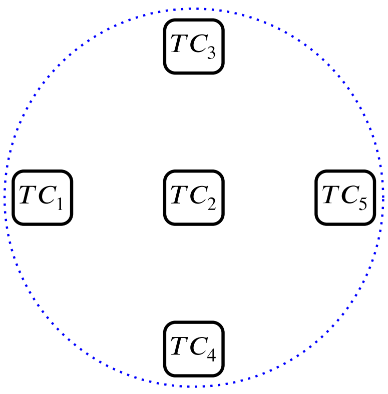

FIGURE 2 Example of an embedded cluster of dependencies for five test cases.

As we can see in Figure 2 , TC_1, TC_3 and TC_4 are independent test cases, TC_2 depends on TC_1. TC_5 directly depends on TC_2, TC_3 and TC_4, where we call them as the precedents, and indirectly depends on TC_1. The presented test cases in Figure 2 can be executed in a different order such as first all independent test cases (TC_1, TC_3 and TC_4) are ranked first for execution, or TC_1 and TC_2 are scheduled for execution first and later two independent test cases TC_3 and TC_4. Detecting this grid of dependencies is required to analyze several layers of the testing process such as requirement specifications and software modules. However, the mentioned required information is not available in all testing projects and sometimes the process of capturing them takes more time than any random test execution. On the other hand, knowing the dependency between test cases can be utilized for other test optimization purposes such as parallel test execution and test automation, where independent test cases can be considered as good candidates for automation in a semi-automated testing procedure. In summary, using the presented information in Figure 1 in total of 328 test cases are detected as independent test cases where the total number of 1420 test cases are dependent test cases.

References

1. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 2008; 14(2): 131.

2. Tahvili S, Ahlberg M, Fornander E, et al. Functional Dependency Detection for Integration Test Cases. In: ; 2018.

3. Arlt S, Morciniec T, Podelski A, Wagner S. If A Fails, Can B Still Succeed? Inferring Dependencies between Test Results in Automotive System Testing. In: ; 2015.

4. Tahvili S, Saadatmand M, Larsson S, Afzal W, Bohlin M, Sundmark D. Dynamic Integration Test Selection Based on Test Case Dependencies. In: ; 2016.

□

[1]It is a dummy in the sense that the number of test cases is very small compared with real industry cases.