Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 2. pp. 273–281
doi: 10.14794/ICAI.9.2014.2.273

Advanced TTCN-3 Test Suite validation with Titan

Attila Kovács[a], Kristóf Szabados[b*]

[a]Eötvös Loránd University
Attila.Kovacs@compalg.inf.elte.hu

[b]Eötvös Loránd University, Ericsson Hungary ltd
Kristof.Szabados@ericsson.com

Abstract

As the size and complexity of large software systems continuously grow, so do their test systems. In today's telecommunication world, we have test systems which are comparable in complexity to that of the tested systems.

Some of these test systems have to simulate millions of users, be as robust as the tested systems themselves and provide comparable performance. To be able to handle such testing needs, ETSI[1] developed the TTCN-3 programming language. TTCN-3 has proved to be very efficient for developing test systems for communication systems.

In this article we present the results of analyzing the test software systems publicly available from www.ttcn-3.org. We show the issues that we have found on semantic level and on advanced code smell level. We also present some results on the test systems structural level.

Keywords: ETSI, 3GPP, TTCN-3

MSC: AMS classification numbers

1. Introduction

In our fast changing world the usage of electrical devices belongs to the everyday life of the society. These devices contain software helping the navigation to destinations, supporting the communication with other people, driving the production, distribution and consumption of energy resources. Software drives companies, trades on the markets, takes care of people's health.

Before Y2K, tests were mostly designed and executed manually. Nowadays every corporation aims at automating their tests which produces large scale test

*Corresponding author
[1]European Telecommunication Standardization Institute

architectures. In the telecom area this pressure facilitated the ETSI to develop a scripting language used in conformance testing of communicating systems and a specification of test infrastructure interfaces that glue abstract test scripts with concrete communication environments. This programming language standard is called TTCN-3 and offers potentials for reducing test development costs significantly.

We look at tests as software products. In this work TTCN-3 is viewed as a programming language. We analyze software products written in TTCN-3 to see how they fulfill quality requirements by applying quality metrics.

For our analysis we extended the Titan tool. Titan is a TTCN-3 test toolset used in Ericsson for functional and load testing with more than 4000 users and freely available for universities, researchers and standardization bodies.

It has been already proven that test systems written in TTCN-3 can be not only large in size, but could also be very complex [5]. The importation graph of these software systems shows scale-free properties. This was one of our main motivations to study the TTCN-3 language and systems in more detail.

The paper is organized as follows. In section 2 we present the projects we have analyzed. In section 3 we present the findings of our low level syntactical and semantical analysis. In section 4 we present our experiences with measuring code smells on the projects. In section 5 we present our findings on structural issues. Section 6 shows how the projects can be clearly separated in their size. Section 7 summarizes the results of this paper.

2. The analysed Projects

We analyzed all test software systems which were available at www.ttcn-3.org, during January 2014. The webpage lists links to test suites provided by 2 different standardization organizations: ETSI and 3GPP[2]. The projects provided by ETSI are:

- WiMax (802.16) Test Suites
- ePassport Readers Interoperability Test Suite
- Session Initiation Protocol (SIP) Test Suite
- IP Multimedia Subsystem (IMS) Test Suites
- IPv6 Test Suites
- Digital Private Mobile Radio (dPMR) Test Suite
- Digital Mobile Radio (DMR) Test Suite
- Intelligent Transport Systems (ITS) Test Suites

The projects provided by 3GPP are:

[2] 3rd Generation Partnership Project

- 3GPP EUTRA (LTE/EPC) UE Test Suites
- 3GPP IMS UE Test Suites
- 3GPP UTRA UE Test Suites
- 3GPP UE Positioning Test Suites

Most test suites had several parts and some even several versions. We decided to measure all software packages, which were available and contained all source files needed to be able to analyze the project. We measured 40 different packages of test suites.

3. Low level findings

We have identified 32 different kinds of syntactical and semantical issues in the examined projects. With the interesting notion, that only ETSI projects contained syntactical errors. None of the 3GPP projects checked contained such low level issues.

3.1. Syntactic issues

We were surprised to find syntactical errors in ETSI testsuites. ETSI is the developer of the TTCN-3 language and these freely available software packages most probably have promotional purposes. We have also noticed that each syntactic error can be traced back to a support tool, which means that the tool vendor misunderstood the standard slightly.

An example for this situation is related to how the brackets of formal parameter lists can be used. According to the TTCN-3 standard[3]: if a "template" structure has no formal parameters, the brackets are not allowed to be written out. The BNF dictates:[3]

```
BaseTemplate ::= (Type | Signature)
    TemplateIdentifier ["(" TemplateFormalParList ")"]
TemplateFormalParList ::= TemplateFormalPar
    {"," TemplateFormalPar}
```

In the available projects we have found cases where these empty formal parameter list brackets were present. An example code is:[4]

```
template ServiceOpt m_serviceOptDefault () := {
emergency := c_emergencyNone,
privacy := c_privacyZero, ...
}
```

On the other hand, as this kind of notation may also make sense, we can imagine that some tool vendor supports it.

[3]TTCN-3 standard [3]: Section A.1.6.1.3

[4]Digital Mobile Radio (DMR) Test Suite; in file DMR_Templates.ttcn lines 16

3.2. Semantic issues

To continue our analysis we temporarily fixed the syntactic problems in our lab environment and analyzed the code semantically. This analysis also brought up several issues:

- In some cases we have found assignments in wrong order. For example in the following code the first field of the structure is filled out 3 times [5].

```
template NbrAdvOptions m_nbrAdvOpt_macTlla3
  (template Oct6to15p_macTlla) := {
  tqtLinkLayerAddr := m_macTlla(p_macTlla),
  tqtLinkLayerAddr := m_macTlla(p_macTlla),
  tqtLinkLayerAddr := m_macTlla(p_macTlla),
  otherOption := omit
}
```

- We also found cases of sub-type restriction violations [6].

```
Bit3 ::= BIT STRING (SIZE(3))
...
const Bit3 c_ackNone := '0'B;
const Bit3 c_ack := '1'B;
```

- We found illegal characters in conversion operations that would drive the test to Dynamic Testcase Error at first execution [7].

```
str2oct("SCOP/1.0");
```

- One of the project sets even has an extension of importing from a proprietary file format [8]. This way the test suite can only be used with one vendor's tool.

3.3. Validation process

We have contacted ETSI in order to provide us with information on why we could find so many issues in the publicly available testsuites. They were kind enough to direct us to the validation manual ([4]) used by ETSI. Section B.2 of this document describes the validation levels that ETSI uses for its products:

1. Basic: The test suite had been compiled on atleast one TTCN-3 tool. Executing the test is not required.

[5] IPv6 Test Suites; TS 102 351 Methodology and Framework; in file LibIpv2_Rfc2461NeighborDiscovery_Templates_PK.ttcn in line 442

[6] Digital Mobile Radio (DMR) Test Suite; type is defined in file CommonLibDataStringTypes.asn line 30; constants in file DMR_Values.ttcn lines 254-255

[7] IPv6 Test Suites; IPv6 Mobility; TS 102 596 version 1.1.1; in file EtsiLibrary/LibScop/LibScop_Codec.ttcn in line 29; fixed in version 1.2.0

[8] IP Multimedia Subsystem (IMS) Test Suites; Netowkr Integration Testing between SIP and ISDN/PSTN; Part4; in file LibSip/LibSip_XMLTypes.ttcn in line 32

2. Strong: The test suite had been compiled on atleast one TTCN-3 tool and executed against atleast one SUT (System Under Test). Running to completion is not required and traces might not be analyzed.

3. Rigorous: The test suite must be compiled on more than one TTCN-3 tool and executed on several test platforms. The complete test suite is executed against SUTs from different suppliers. The operation and output of the tests have been validated.

According to this information our findings shows that the publicly available test suites were not validated on level 3.

We tried to check this information but we could not find any clear reference. 1) The project web-pages do not list this information, 2) the documents attached to these projects contain only formal descriptions (naming conventions, architectural descriptions, etc.), 3) most of the packages, containing the source codes, have no non-source code files at all.

It is also mentioned that the Technical Committee of any given Test Suite has the responsibility to decide which validation level to use. This can result in high diversity in quality among the Test Suites.

4. Code smells

We used code smells (defined in [6]) to measure the software quality of test suites.

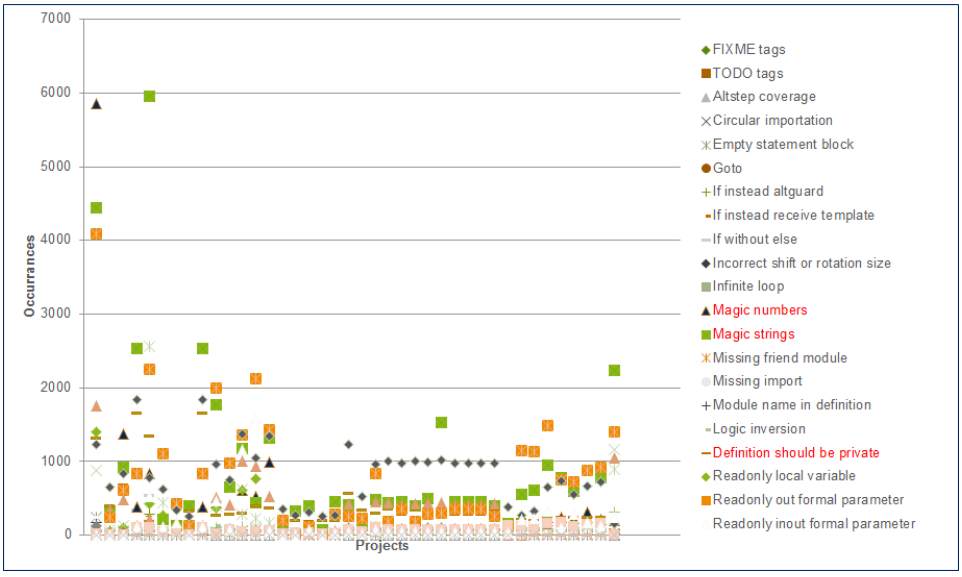

Figure 1: Code smells measured on the projects

Figure 1 shows our findings. Although the amount of code smells we have found differs in each project, the frequency of the smells are relatively the same.

The top 4 code smells occurring most in every project are:

- Magic strings and numbers,
- Un-initialized local variables,
- Unused global definitions,
- Definitions that could be private, but are not set so.

Some of these come from the original idea behind the language: let writing testcases be as easy and as fast as possible.

TTCN-3 supports a compact representation of values, enabling high development speed. This also helps burning "magical" values into the source code, which can lead to understandability and changeability problems.

Unused global definitions might mean: 1) there are still functionalities for which there are no tests, 2) some parts of the system are not needed and overcomplicate the system without adding benefits.

Unused local variables might point out implementation issues: either the implementer was not careful enough to not leave behind unnecessary code, or the unused variable was planned to be used, which might mean incorrect behavior.

The idea of visibility was not present in the language for several years: the first version of the standard appeared in 2001-01 [1], visibility attributes for definitions were added in 2009-03 [2], while the current version appeared in 2013-04 [3]. Having every type, data and functionality publicly available speeds up the writing of tests, but in the long run this practice can create hard to maintain architectures. Internal representation cannot change after customers started using it, without imposing extra costs on the customers side.

5. Structural issues

We studied the structure of the available projects as well and visualized their module importations as graphs. On these graphs each node is a module of the project (a single compilation entity). Edges are directed connections which represent a single import. Nodes without incoming edges are moved to the top to create a layered structure. This visualization for example allowed us to spot modules which are not connected to the rest of the graph. When we checked the structure of the ETSI projects we found such nodes. Figure 2 shows that the modules "SipIsdn_PICS", "LibCommon_Time" and "General_Types" are not connected to the rest of the structure. This effect was also seen in the project "Digital Private Mobile Radio".

We note that at this level we were no longer able to check every ETSI project as we could not easily recover the code from syntactical and semantical issues. On the other hand we were able to analyze all 3GPP projects (figure 3 shows EUTRA). According to our analysis none of them had loose modules.

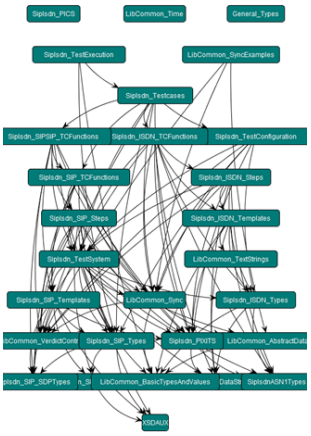

Figure 2: Module importation graph of IMS interworking testsuite.

6. Relations to size

We have measured the size of these projects to see if there is a difference in what ETSI and 3GPP works with. We have found that the number of modules of the 3GPP projects were between 56 and 249; while the depths of the DAG (Directed Acyclic Graph) have 15 to 18 levels. ETSI projects have 8 to 68 modules and the depths of the DAG have 5 to 15 levels.

There seems to be a clear separation in size between the projects of the two organizations. 3GPP is working with projects having much more modules and larger network structure.

We also measured the cumulative project risk factors that were defined in [6]. Figure 4 shows our findings. According to our measurements the average project risk factor turned out to be 60.075 points. In this case there was no big difference between ETSI and 3GPP developed test suites. The 3 projects with the lowest risk factors are all part of the Intelligent Transport Systems test suites developed by ETSI.

7. Summary

We have analyzed all software packages that were available at www.ttcn-3.org. We have found that although ETSI is the developer of the TTCN-3 language (in which these software are written) some of their projects contain even syntactical issues. During our analysis we have found syntactical, semantical and even structural problems in ETSI provided software packages.

Our code smell analysis shows that there would be a need for an automatic code

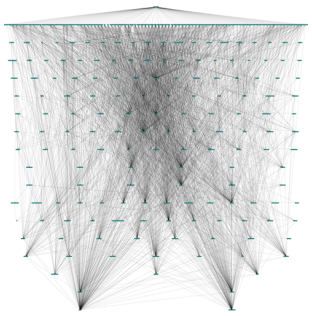

Figure 3: Module importation graph of Evolved Universal Terrestrial Radio Access.

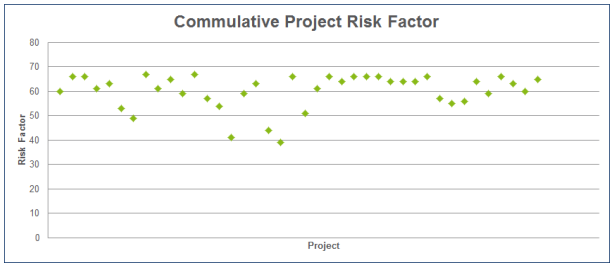

Figure 4: Title of this picture

smell analyzer. Currently there might be several quality issues which can turn out to be bugs. Unclear or too permissive API definitions can lead to problems once these test suites are inserted into industrial systems and are built upon or extended.

According to our findings currently the 3GPP provided test suites are of higher software quality than the ETSI provided ones.

We have taken up contacts with ETSI representatives and notified them about the issues we have found. We received a promise that they will start a project in the near future to correct the problems.

Acknowledgements. The authors would like to thank the DUCN Software Technology unit of Ericsson AB, Sweden for the financial support of this research and the Test Competence Center of Ericsson Hungary to provide access to their in-house tools. These proved to be invaluable to measure our data.

We would also like to thank Gábor Jenei, Dániel Poroszkai and Dániel Góbor for their help in implementing features that were crucial to our investigation. Their work allowed us to quickly process large amount of data.

References

[1] European Telecommunications Standards Institute Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language Version 1.0.10 *"http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/01.00.10_50/es_20187301v010010m.pdf"*

[2] European Telecommunications Standards Institute Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language Version 4.1.1 *"http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.01.01_50/es_20187301v040101m.pdff"*

[3] European Telecommunications Standards Institute Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language Version 4.5.1 *"http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.05.01_60/es_20187301v040501p.pdf"*

[4] European Telecommunications Standards Institute ETSI EG 201 015 V2.1.1 Methods for Testing and Specification (MTS); Standards engineering process; A Handbook of validation methods *"http://www.etsi.org/deliver/etsi_eg/201000_201099/201015/02.01.01_60/eg_201015v020101p.pdf"*

[5] K. Szabados Structural analysis of large TTCN-3 projects, Proc. 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop, *Lecture Notes in Computer Science 5826:, Testing of Software and Communication Systems*, Springer-Verlag Berlin, Heidelberg, 2009 pp. 241246.

[6] A. Kovács, K. Szabados Test software quality issues and connections to international standards, *Acta Univ. Sapientiae, Informatica*, 5, 1 (2013) 77102 http://www.acta.sapientia.ro/acta-info/C5-1/info51-6.pdf