TRex – The Refactoring and Metrics Tool for TTCN-3 Test Specifications

Paul Baker, Dominic Evans
Motorola Labs, Jays Close, Viables Industrial Estate,
Basingstoke, Hampshire, RG22 4PD, UK
{paul.baker,vnsd001}@motorola.com

Jens Grabowski, Helmut Neukirchen, Benjamin Zeiss
Software Engineering for Distributed Systems Group,
Institute for Informatics, University of Göttingen,
Lotzestr. 16-18, D-37083 Göttingen, Germany.
{grabowski,neukirchen,zeiss}@cs.uni-goettingen.de

Abstract

Comprehensive testing of modern communication systems often requires large and complex test suites which then have to be maintained throughout the system life-cycle. Industrial experience, with those written in the standardised Testing and Test Control Notation *(TTCN-3), has shown that this maintenance is a non-trivial task and its burden could be reduced if appropriate tool support existed. To this aim, Motorola has collaborated with the University of Göttingen to develop TRex, a TTCN-3 development environment published under the Eclipse Public License, which notably provides suitable metrics and refactorings to enable the assessment and automatic restructuring of test suites. In this paper we present the TRex tool, which will make it far easier to construct and maintain TTCN-3 tests that are concise and optimally balanced with respect to readability, usability, and maintainability.*

1. Introduction

The *Testing and Test Control Notation* (TTCN-3) [3, 5] is a test specification and test implementation language standardised by the *European Telecommunications Standards Institute* (ETSI) and the *International Telecommunication Union* (ITU). While TTCN-3 has its roots in functional black-box testing of telecommunication systems, it is nowadays also used in other domains such as Internet protocols, automotive, aerospace, or service-oriented architectures. TTCN-3 can be used not only for specifying and implementing functional tests, but also for scalability, robustness or stress tests. TTCN-3 is based on a textual core notation and several presentation formats.

Commercial TTCN-3 tools [7, 8, 11] support editing test suites and compiling them into executable code. By implementing the interfaces of the standardised TTCN-3 runtime environment, these tools may also allow TTCN-3 test campaigns to be managed and executed.

However, experience within Motorola has shown that not only editing and execution of TTCN-3 test suites, but also maintenance and assessment of these are important issues which require tool support [1]. It is not always obvious how to use TTCN-3 concepts such as templates in a manner that can maximise readability, usability and maintainability, so user guidance and assistance in these areas would be extremely beneficial. In addition, Motorola teams have encountered related problems whilst migrating legacy test suites to TTCN-3; the conversion of tests for a UMTS based component to TTCN-3 initially resulted in 60,000 lines of code which were hard to read, hard to (re-)use, and hard to maintain, as the benefits of moving to TTCN-3 were not yet being fully exploited. In legacy modelling languages the specification of data values is often tightly coupled with the behaviour specification, i.e. when a signal is sent between processes, the value of the signal is defined within the behaviour of each process. This means that when the signal type is modified, each behaviour definition must also be modified and value definitions cannot easily be reused. In this case the task of manually decoupling data from behaviour to improve maintainability, whilst also reducing suite size by merging commonalities and hence improving readability, proved to be particularly challenging and time-consuming.

Currently, no tools for assessing and improving the quality of TTCN-3 test suites exist. To this end, Motorola has collaborated with the University of Göttingen to develop a TTCN-3 refactoring and metrics tool, called *TRex*. The initial aims of TRex were to: (1) enable the assessment of a TTCN-3 test suite with respect to lessons learnt from experience, (2) provide a means of detecting opportunities to avoid any issues, and (3) a means for restructuring TTCN-3 test suites to improve them with respect to any existing issues. To let others participate in our tool and to participate in contributions from others, we have made TRex a general open-source quality assurance and quality improvement tool for TTCN-3 test suites.

This paper is structured as follows: In the next section, we will give an overview of TRex's functionality following with a description of the implementation in Section 3. In Section 4 we conclude with a summary and outlook.

2. Functionality of the TRex tool

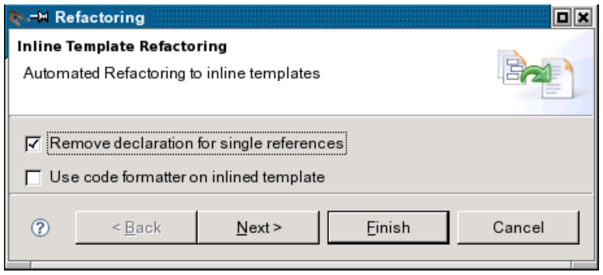
TRex is implemented as an Eclipse plug-in and therefore, anyone who has had experience with the Eclipse Platform [2], e.g. by using the popular *Java Development Tools* (JDT), will immediately feel comfortable with TRex. The TTCN-3 perspective of TRex allows editing of TTCN-3 core notation as well as assessing and improving the quality of TTCN-3 test suites.

2.1. Editing

TRex provides editing facilities known from a typical *Integrated Development Environment* (IDE). These include a *Navigator* view for project browsing, an editor with syntax highlighting and checking according to the latest TTCN-3 core language specification (v3.1.1), an *Outline* view providing a tree representation of the TTCN-3 structure for the currently edited file, *Content Assist* which automatically completes identifiers from their prefix and scope, a code formatter, a reference finder which displays all references to a given element, and the possibility to jump to the declaration of a given reference. In addition, to allow the edited tests to be compiled and run against either a real or emulated system under test, the Telelogic Tau G2/Tester [8] analyser and compiler *t3cg* is integrated into the TRex environment.

2.2. Refactoring

As a powerful means for improving the quality of TTCN-3 test suites, TRex is able to restructure test suites in an automated way. This is achieved by using *refactoring* which is defined as "*a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior*" [4]. While refactoring is well known for implementation languages like Java, it has not been systematically studied for TTCN-3. Thus, we developed a catalogue of 49 refactorings applicable for TTCN-3 [13, 14]. In TRex, we have begun


Refactoring

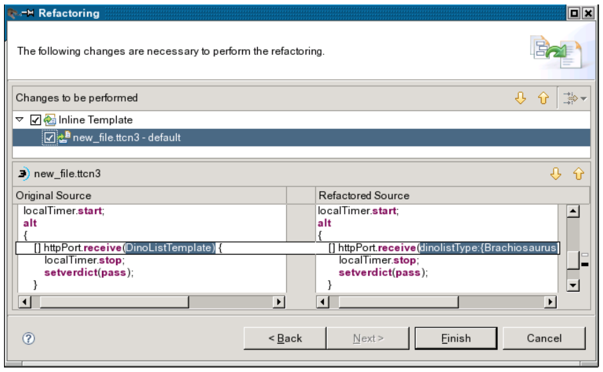
Inline Template Refactoring

Automated Refactoring to inline templates

[✓] Remove declaration for single references

[ ] Use code formatter on inlined template

(a) Configuration


Refactoring

The following changes are necessary to perform the refactoring.

Changes to be performed

[✓] Inline Template

[✓] new_file.ttcn3 - default

new_file.ttcn3

Original Source

```
localTimer.start;
alt
{
  [] httpPort.receive(DinoListTemplate) {
    localTimer.stop;
    setverdict(pass);
  }
```

Refactored Source

```
localTimer.start;
alt
{
  [] httpPort.receive(dinolistType:{Brachiosaurus
    localTimer.stop;
    setverdict(pass);
  }
```

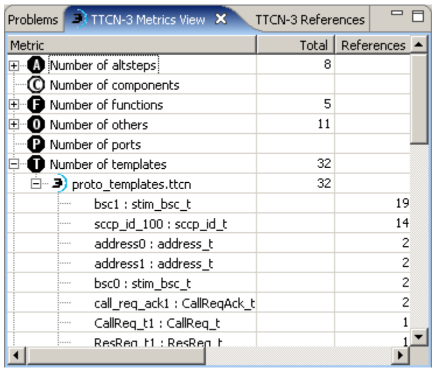
(b) Preview

Figure 1. Wizards for the *Inline Template* refactoring

implementing those refactorings which we believe would improve the maintainability of Motorola's test suites and have so far completed the *Inline Template*, *Inline Template Parameter*, *Merge Templates*, and *Rename* refactorings. As an example, we will describe the *Inline Template* refactoring in detail.

For specifying test data, TTCN-3 uses so called *templates*. A template may either be defined as a named entity on its own or "on-the-fly" using an *inline template* notation. The first way promotes re-use since a template definition may be referenced at several locations. In contrast, test behaviour may be easier to understand if it uses inline templates, since inline templates define test data at the same location where it is actually used for sending or receiving.

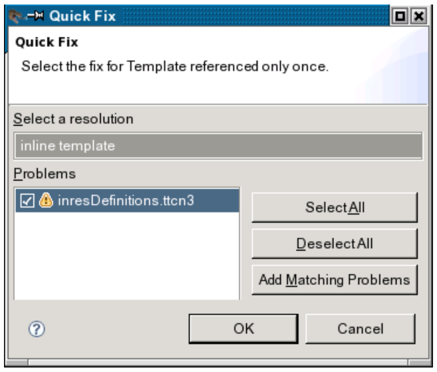
The *Inline Template* refactoring allows a template reference to be transformed into its semantically equivalent inline template notation. The application of this refactoring is particularly reasonable if a template is only referenced once. When applying this refactoring to a template reference, TRex opens a wizard dialogue which offers configuration for the *Inline Template* refactoring. As shown in Figure 1(a), it is possible to remove the declaration of a template if it was referenced only once and the Formatter may additionally be used to obtain a pretty-printed template. Before a refactoring is actually applied, the refactoring wizard displays a preview of all resulting changes (Figure 1(b)).



Problems | TTCN-3 Metrics View | TTCN-3 References

Metric	Total	References
Number of altsteps	8	
Number of components		
Number of functions	5	
Number of others	11	
Number of ports		
Number of templates	32	
proto_templates.ttcn	32	
bsc1 : stim_bsc_t		19
sccp_id_100 : sccp_id_t		14
address0 : address_t		2
address1 : address_t		2
bsc0 : stim_bsc_t		2
call_req_ack1 : CallReqAck_t		2
CallReq_t1 : CallReq_t		1
ResReq_t1 : ResReq_t		1

(a) TTCN-3 Metrics view



Quick Fix

Quick Fix

Select the fix for Template referenced only once.

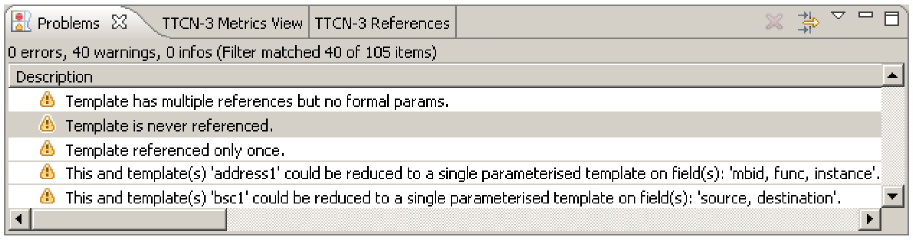
Select a resolution

inline template

Problems

☑ inresDefinitions.ttcn3

(b) TRex's Quick Fix suggestion



Problems | TTCN-3 Metrics View | TTCN-3 References

0 errors, 40 warnings, 0 infos (Filter matched 40 of 105 items)

Description
Template has multiple references but no formal params.
Template is never referenced.
Template referenced only once.
This and template(s) 'address1' could be reduced to a single parameterised template on field(s): 'mbid, func, instance'.
This and template(s) 'bsc1' could be reduced to a single parameterised template on field(s): 'source, destination'.

(c) TRex's rule-based refactoring suggestions

Figure 2. TRex's Metrics-based functionality

These refactorings are typically semi-automated, since the user still has to identify locations where they should be applied (as known from JDT for example). However, as shown in the next section, TRex can also automatically identify such locations.

2.3. Metrics

As part of TRex we are investigating the application of metrics to give an indication of both the overall quality of a TTCN-3 test suite and any locations where it might be beneficial to apply a particular refactoring.

We have implemented basic size metrics in the TRex tool, including *Number of...* and *References to ...* for various definitions and types as well as a measure, labelled *Template coupling*, of the dependency between test behaviour and test data (in the form of template definitions). Figure 2(a) shows the *TTCN-3 Metrics* view.

Based on these metrics, we have defined several rules by which the templates of a TTCN-3 test suite can be analysed, e.g. looking at number of references, use of parameters, commonalities, etc. From these, TRex is able to identify problematic code fragments and to suggest suitable refactorings. For example, templates which could be removed, inlined, or merged into a common parametrised version. These suggestions are displayed in the *Problems* view as warnings (Figure 2(c)) and can either be treated merely as indicators that should be taken into account whilst working on the test suite, or an associated *Quick Fix* can be invoked via the context menu to perform a suggested refactoring automatically (Figure 2(b)). A full description of our metrics and rules is available in our previous paper on TTCN-3 refactoring [14].

3. Implementation of the TRex tool

Building an IDE on Eclipse is attractive from the developer's point of view as it is well documented and supported, and provides many ready-to-use components. Such components include project and file management (workspace) and a graphical user interface (workbench) which can be configured to match the typical layout of an IDE. In fact, the majority of TRex's functionality is built upon abstract implementations provided by Eclipse.

Figure 3 shows the TRex tool chain: the Eclipse Platform provides the basic IDE infrastructure. The TRex components build on top of the Eclipse Platform. They are explained in the subsequent sections.

3.1. Static analysis

The foundation for most functionality in TRex is the TTCN-3 parser and the resulting syntax tree[1]. For building up the syntax tree for a test suite we use *'ANother Tool*

[1]An alternative approach would be to build up a TTCN-3 meta model [10] representation of a TTCN-3 test suite and to use this representation instead of the syntax tree.

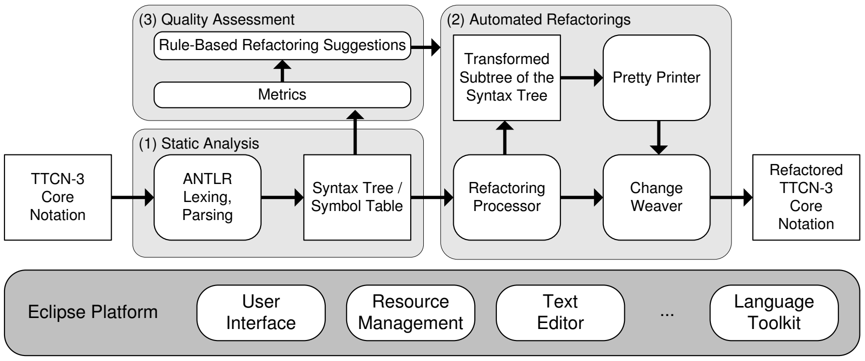


Figure 3. The TRex tool chain

for Language Recognition' (ANTLR) [9], a parser generator which supports lexing, parsing, and syntax tree creation and traversal. For tree traversal, ANTLR uses tree grammars (e.g. the pretty printer uses a tree grammar enriched with semantic actions for the syntax reconstruction).

Most of the advanced functionality of TRex requires additional information for TTCN-3 identifiers, such as the identifier's type, or the syntax tree node of its declaration. To easily find this information, a symbol table was implemented. The syntax tree and the symbol table provide the basis upon which most of TRex's present functionality is realised, e.g. the metrics and refactoring implementations both use them. As shown in Block (1) of Figure 3, the lexer creates a token stream from the TTCN-3 core notation which is used by the parser for syntactical validation and for building the syntax tree. In addition, the symbol table is also created here.

3.2. The refactoring implementation

The refactoring implementations make use of the Eclipse *Language Toolkit* (LTK) which provides abstract classes for semantic preserving workspace transformations and customisable wizard pages for the user interaction. The benefit of such wizard pages is, for example, an integrated preview pane that can be used to compare the original source to the refactored source side by side.

Block (2) in Figure 3 depicts how the automated refactorings are realised. On the basis of the static analysis step (Block (1)), the workspace transformations can be calculated once the concerned syntax tree node (or nodes respectively) has been found through a data structure which stores identifiers along with their text file offsets and once the user entered any required information in the refactoring wizard.

The transformation of the workspace resources (i.e. text files) is realised with a programmatic text editor provided by the Eclipse Platform. It supports copy, paste, move, delete, insert, and replace operations. These operations are used to weave only the textually changed parts into the original TTCN-3 source files. Therefore most of the original formatting is preserved. In some cases an intermediate step involving a syntax tree transformation may become necessary, in order to calculate the required changes. In this case, the TTCN-3 core notation to be weaved into the original TTCN-3 source files is obtained by the pretty printer. Applying multiple changes to a single file is supported by the programmatic editor by automatically tracking changing offset positions.

3.3. Metrics and refactoring suggestions

Metrics are measured immediately after the syntax tree for a test suite has been built or updated (Block (3) in Figure 3). The tree is then fully traversed; all definitions that metrics will be calculated for (*altstep*, *function*, *template*, etc.) are recorded and all communication statements (*send*, *receive*, etc.) are processed to derive *Template coupling* scorings. References to all of these are calculated in a further pass of the tree, hence giving enough information for the basic size metrics (described in Section 2.3) to be displayed. Then all templates found in the first step are processed one-by-one against the analysis rules, using the previously calculated referencing information as well as further inspection of their structure.

Once this has completed, the rule findings are associated with the templates in the form of customised Eclipse *marker* objects which are automatically displayed in the *Problems* view. *Quick Fixes* are resolved for each of them based on extended attributes which indicate the detected situation and hence some corresponding suggestion(s) from the rule set.

4. Summary and outlook

We presented TRex, an Eclipse-based TTCN-3 development environment with an emphasis on quality assessment and quality improvement of TTCN-3 test suites. TRex has been developed as a collaboration between Motorola and the University of Göttingen to address industrial demands by applying current research results. Besides editing functionality, TRex provides user-initiated semi-automated refactoring of TTCN-3 test suites, as well as fully automated refactoring based on special rules which interpret metric values.

Future versions of TRex will include enhanced editing capabilities and further metrics, refactoring, and analyses for TTCN-3 test suites. Therefore, we have started to implement control-flow- and call-graphs for TTCN-3 behaviour. These graphs will be used, for example, to provide complexity metrics and to allow the detection of anomalies in the control- and data-flow.

Even though TRex is still under development, we have already started our first experiments into the use of TRex for quality improvement and assessment. The results are very promising. For example, test suites consisting of many templates were reduced in size considerably by automatically removing unused templates and by merging similar templates into common parametrised versions. We have also started to analyse existing real-world TTCN-3 test suites in order to determine appropriate boundary values for our metrics. These boundary values are very important for the optimal interpretation of calculated metrics.

The use of metrics to assess the quality of TTCN-3 test suites and to suggest appropriate refactorings is only one possible approach. A further approach, which we would like to pursue in the future, is to identify anti-patterns, i.e. inappropriate usage of TTCN-3 (so called "bad smells"). In contrast to the calculation of metrics, this requires a pattern-based approach, e.g. to identify duplicate code.

TRex is an open-source tool that is freely available under the Eclipse Public License at its website [12]. In addition to the application of TRex in industry, TRex is also being used as a tool for academic research and teaching. We invite the TTCN-3 community to use the tool, share their experience, and participate in the future development of TRex.

Currently, we have defined the notion of quality of TTCN-3 test suites only in an informal manner: e.g. using inline templates reduces maintainability but improves readability (but only up to a certain size of a template) or parametrised templates promote reuse, but not necessarily maintainability or readability. However, as a solid foundation for test suite assessment and improvement a more formal quality model for test suites is desirable. For quality of software products in general, the *International Organization for Standardization* and the *International Electrotechnical Commission* have defined quality characteristics (functionality, reliability, usability, efficiency, maintainability, and portability), sub-characteristics, and metrics for measuring attributes of these characteristics [6]. Thus, for future work the challenge remains to investigate the applicability of this existing quality model to test specifications.

References

[1] P. Baker, S. Loh, and F. Weil. Model-Driven Engineering in a Large Industrial Context – Motorola Case Study. In L. Briand and C. Williams, editors, *Model Driven Engineering Languages and Systems: 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005*, volume 3713 of *Lecture Notes in Computer Science (LNCS)*, pages 476–491. Springer, May 2005.

[2] Eclipse Foundation. Eclipse. `http://www.eclipse.org`, 2006.

[3] ETSI European Standard (ES) 201 873 V3.1.1 (2005-06): The Testing and Test Control Notation version 3; Parts 1–7. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.140–Z.146, 2005.

[4] M. Fowler. *Refactoring – Improving the Design of Existing Code*. Addison-Wesley, 1999.

[5] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles, and C. Willcock. An Introduction into the Testing and Test Control Notation (TTCN-3). *Computer Networks*, 42(3), June 2003.

[6] ISO/IEC Standard No. 9126: Software engineering – Product quality; Parts 1–4. International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC), Geneva, Switzerland, 2001-2004.

[7] OpenTTCN Oy. OpenTTCN Tester for TTCN-3. `http://www.openttcn.com/Sections/Products/OpenTTCN3`, 2006.

[8] Telelogic AB. Tau/Tester. `http://www.telelogic.com/corp/products/tau/tester/index.cfm`, 2006.

[9] T. Parr. ANTLR parser generator. `http://www.antlr.org`, 2006.

[10] I. Schieferdecker and G. Din. A Meta-model for TTCN-3. In M. Núñez, Z. Maamar, F. Pelayo, K. Pousttchi, and F. Rubio, editors, *Applying Formal Methods: Testing, Performance and M/ECommerce, FORTE 2004 Workshops, Toledo, Spain, October 1-2, 2004*, volume 3236 of *Lecture Notes in Computer Science (LNCS)*, pages 366–379. Springer, 2004.

[11] TestingTechnologies. TTworkbench. `http://www.testingtech.de/products/ttwb_intro.php`, 2006.

[12] TRex Website. `http://www.trex.informatik.uni-goettingen.de`, 2006.

[13] B. Zeiss. A Refactoring Tool for TTCN-3. Master's thesis, Institute for Informatics, University of Göttingen, Germany, ZFI-BM-2006-05, Mar. 2006.

[14] B. Zeiss, H. Neukirchen, J. Grabowski, D. Evans, and P. Baker. Refactoring for TTCN-3 Test Suites. In *Proceedings of SAM'06: Fifth Workshop on System Analysis and Modelling, May 31–June 2, 2006, University of Kaiserslautern, Germany*, 2006.