Test Patterns with TTCN-3

Alain Vouffo-Feudjio and Ina Schieferdecker

Fraunhofer FOKUS, Berlin, Germany
{vouffo, schieferdecker}@fokus.fraunhofer.de

Abstract. Patterns are used in various engineering disciplines to represent common aspects of a set of solutions to a (set of) common problem(s). This paper discusses main concepts of test patterns, provides a characterization of test patterns and describes the use of test patterns in the test development process. Test patterns can be used to support the vertical reuse between different testing phases and testing kinds, horizontal reuse between different product version and historical reuse between different product versions. Specification means for test patterns will be analysed and compared with the language features of the Testing and Test Control Notation (TTCN-3) [8].

1 Introduction

Test patterns represent a form of reuse in test development in which the essences of solutions and experiences gathered in testing are extracted and documented so as to enable their application in similar contexts that might arise in the future. Test patterns aim at capturing test design knowledge from past projects in a canonical form, so that future projects would benefit from it. In this paper we will describe different views on the concept of test patterns and discuss methodological aspects such as notation, test pattern mining and test pattern application. The concept of patterns as it is currently known in the software development community originates from the works of Alexander [1], a building architect who had the basic idea of recording design wisdom in a canonical form. He defines a pattern as "both a description of a thing which is alive, and a description of the process which will generate that thing". It soon became obvious that the concept of patterns introduced by Alexander for the buildings architecture domain could also apply to nearly any design and engineering field. As [2] pointed out: "A pattern is the result of abstracting from a given (set of) problem-solution pair(s) and distilling common factors, which can be reused to solve other problems".

In analogy to the patterns for urban architecture, software designers acknowledged the existence of patterns in software architecture and the need for identifying and describing them in such a way that they would possibly be reused wherever the context might require it. Therefore, [3] defines a software architecture pattern as both a part of a software system and a description of how to build that part. The purpose of software architecture patterns is to identify and specify abstractions above level of single instances or components in a software system, as well as to document existing well proven design experiences, software architectures and guidelines. Also software patterns provide a common vocabulary and understanding for design principles and well-proven experiences.

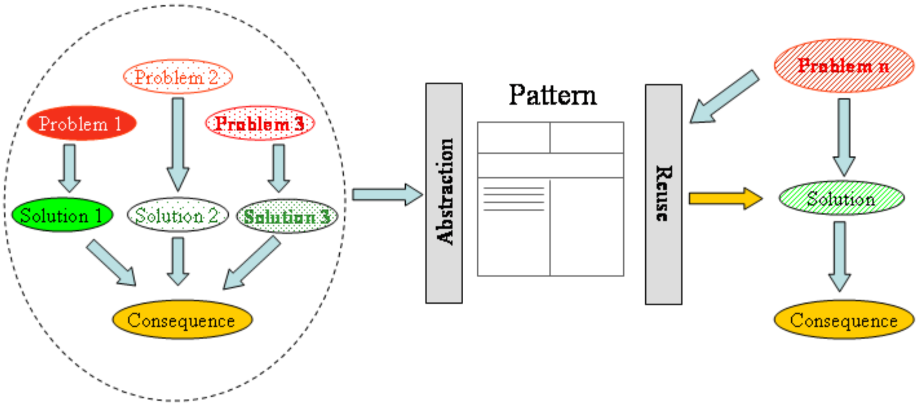

Fig. 1. Pattern Extraction and Reuse: The process of extracting the essence of a set of problem-solution-benefit combinations is displayed in the left hand part of the figure, whereas on the right-hand side, the reverse process of producing a solution for a similar problem-benefit pair by applying the previously identified pattern is illustrated

However some severe debates also emerged around patterns and how they relate to existing software methodologies. As described in [2], emphasis must be put on the fact that patterns can and should not be viewed as solution for all possible software engineering problems and one should not attempt to force patterns reuse in situations where they simply would not fit. Patterns should rather be viewed as a complementary approach to existing methodologies. Also patterns should harmonize with the fundamental principles of software construction commonly known as “enabling techniques”, which are independent of a specific software development method such as Abstraction, Encapsulation, Information Hiding, Modularization, Separation of Concerns, Coupling and Cohesion, Sufficiency, Completeness and Primitiveness, Separation of Policy and Implementation, Separation of Interface and Implementation, Single Point of Reference, and Divide-and-conquer.

While some patterns address some of those concepts explicitly, it is important to make sure that patterns do not affect those principles. This also applies for the usual non-functional requirements on software systems, i.e. changeability, interoperability, efficiency, reliability, testability and reusability. It should be kept in mind, that while some patterns will aim at enhancing some of those requirements and help in achieving them, it is also possible that a given pattern affect some of the non-functional requirements negatively. For example, the broker pattern, which is the base of many middleware architectures such as CORBA, eases testing of individual client or server components in a distributed system. However it decreases the testability of client-server systems by introducing additional elements between the client and the server.

Test systems and more specifically TTCN-3 test systems are a special type of software systems and with their growing complexity, the need for cataloguing good practices with regard to design, architecture, implementation and execution is becoming more and more urgent. Just as for "normal" software, well-proven experiences gathered while developing TTCN-3 based test systems need to be documented to ease their reuse. Therefore we define a test pattern as a software pattern that applies specifically to the testing domain. Similarly to general software system engineering, the benefits expected from patterns in testing are a reduction of time-to-market and costs through reuse of existing test artefacts. It should be pointed out here, that the term "test artefacts" in this context is not limited to actual (TTCN-3) code but also include concepts and principles at a higher level of abstraction.

Reuse of tests can be compared with reuse of software. Test suites and their constituents (like test cases or test data) may be reused as is or may need adaptations before they can be reused. E.g. it is possible to develop parameterized test cases, which enables their adaptation to different testing contexts. Mainly, three different approaches are important for test reuse: the vertical, horizontal and historic reuse. Vertical reuse is on the reuse possibilities between different testing phases such as requirements testing, prototype testing, module testing, integration testing, system testing, and acceptance testing. Another approach is between different types of testing, e.g. tests developed for functional testing could be reused for performance or scalability testing [5]. The horizontal reuse addresses the reuse of tests between various products and within product families. The historical reuse addresses test reuse between product generations. Historical and horizontal reuses are similar as "different products" in horizontal reuse could be considered as "different product versions" in historical reuse. The differences in products/product families and product versions will show in different testing parts being reused for testing.

2 Categories of Test Patterns

The issue of patterns in general and specifically that of test patterns has very often been a source of some misunderstanding among experts. This stems from the generality of the concept and the fact that depending on the aspect of test engineering on which focus is laid, apparently different definitions and classifications might emerge. In our current work we've identified the following views on test patterns:

- The scope-based view: The nature of the test development process may vary a lot, depending on the scope or target of those tests. It is obvious, that the techniques for generating, specifying, executing and evaluating the tests can be totally different, whether unit testing, integration testing or system testing is being performed. Examples of patterns for unit testing have been provided by [11] , whereas some others for component testing of object oriented software as described in [12] and in [7].
- The management-based view: In some cases such as [13], the issues of patterns in test development is discussed at a higher level of abstraction, which involves aspects such as the management of test projects and test organizations, the strategies for achieving higher efficiency in testing etc.

We chose to adopt a scope-based approach and focus on the process of developp-ing a test system based on a formal notation such as TTCN-3 or the UML 2.0 test profile (U2TP). Therefore in the coming section we will go through the different phases of developing a TTCN-3 based test system and discuss which type of test patterns could be identified and possibly reused to ease or improve that phase of the process.

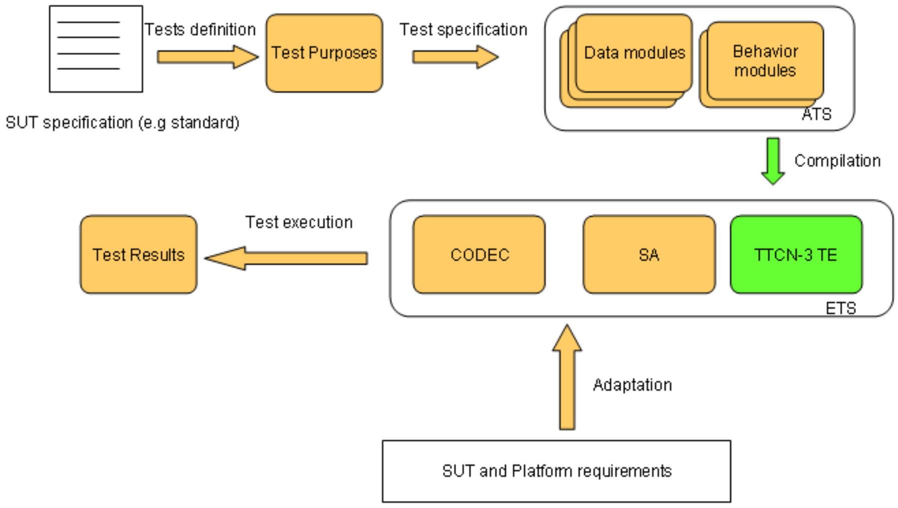

Fig. 2. Above illustrates the process of TTCN-3 based test development, going from a specification of the system under test (SUT) to an executable test suite which can be run to provide the required test results. To obtain a complete executable test suite (ETS), elements related to the SUT and the test platform, such as the encoder/decoder (CODEC) and the system adapter (SA) must be developed and combined to the TTCN-3 test executable (TE) generated automatically from the abstract test suite (ATS) through compilation

2.1 Patterns in the Test Definition Phase

Test definition is the first phase in building a TTCN-3 based test system. It consists of extracting test purposes from the SUT's specification, depending on what the test goals are. Further analysis is required to try to identify possible patterns in this process, for example along certain families of SUTs. Questions are e.g.

- Do generic patterns exist for extracting test purposes from SUT specification?
- Which test purposes are more prone to detect failures in a given SUT domain such as a telecommunication protocol and should therefore always be present in test suites for that domain?
- For SUTs specified in a formal language, could generic patterns for extracting test purposes for such systems be identified and made available for reuse?

With the process of deriving TTCN-3 test cases from test purposes being quite costly and error-prone, the need for formalizing how test purposes are described has arisen. In the Pattern for Test Development (PTD) group setup by the European Telecommunications Standards Institute (ETSI) some patterns have been proposed for that purpose [4]. One long term goal is to enable automatic derivation of test cases from such formalized and machine processable test purposes. One of the purposes of our work will be to analyze, how that approach fits in the current picture of test purpose specification notations and how suitable it would be for automatic generation of test suites (or skeletons).

2.2 Patterns in the Test Specification Phase

The test specification phase deals with the abstract definition of test data and test procedures in TTCN-3 yielding an ATS (abstract test suite). Using some concepts of the TTCN-3 language such as the import mechanism, value parameterization and modifiable templates enables the test developer some support for code reusability. The main elements of an abstract test suite can be separated in a static part and a dynamic part:

- The static part is also referred to as test structure or test model and describes a model of the test suite containing
  - A specification of the types of components to be involved in the test suite and the ports they provide.
  - A specification of the data needed for communication between the components and with the system under test (SUT).
- The dynamic part specifies
  - how test components of the types specified in the static part are created and connected with each other or the SUT to build a test configuration or a test architecture, and
  - the message flow between the elements of the test architecture and the mechanisms for verdict assignment for each test case.

Following the path above, TTCN-3 test patterns can be classified in 4 basic categories depending on their application area:

- Architectural patterns
  Architectural patterns address solutions as to how test systems could be configured to solve or avoid specific recurring problems and to ensure that the fundamental principles mentioned in the introduction to this document are not jeopardized. This also includes patterns for the coordination and synchronization of test components in a test system.
- Behavioural patterns
  Behavioural patterns provide ways for defining the behaviour of elements in a test suite. Behaviour patterns might apply for single elements of that test suite, as well as for the interaction of test components with each other or with a given SUT.

- Data patterns
  Data patterns are test patterns describing reusable concepts and approaches for specifying and generating test data.
- Test reuse patterns
  Test reuse patterns describe strategies for reusing existing TTCN-3 tests
  - for evaluating different aspects of the SUT. E.g. proven approaches for using tests originally defined for conformance or functional testing to test the SUT with regard to load management (scalability) or performance, or
  - along the SUT's product lifecycle, i.e. how to ensure that major modifications on the test suite would not be required as new versions of the SUT are developed.

2.3 Patterns in the Test Adaptation Phase

After completing the abstract specification of the test suite, the test adaptation phase of the test development process starts. In this phase, the ATS is extended with the test runtime interface (TRI) [9] and test communication interfaces (TCI) [10] adaptors to build the fully executable test suite (ETS) which can actually be run to assess the SUT's correctness. Typically, the system adaptor (SA), the platform adaptor (PA) and the codec (CD) are to be modified. Some of the questions which need to be answered in that phase are:

- What are the patterns for encoders/decoders for TTCN-3 test systems?
- What are the patterns to test encoders/decoders on proper functioning?
- What are the Do's and Don'ts to be followed when writing a TTCN-3 system adapter?
- Which well proven techniques and experiences can one rely on in that process?

2.4 Patterns in Test Execution

In the test execution phase, two key aspects are to be considered for possible patterns. Test management patterns cover solutions for ensuring that test suites are executed in a way that their results are relevant for the SUT in that they enable error detection on the system and provide the person executing the tests with means for identifying which parts of the system does not function as required. This includes tracing and logging approaches during test execution, avoiding memory leaks during test execution etc. Test execution automation patterns focus on methods for effectively automating the test execution process, independently of the scale of the test suite.

3 Methodological Aspect of Test Patterns

A methodology for test patterns should not only address the various kinds of test patterns for the different approaches of test reuse, but also define how a test pattern is to be defined, identified, selected and applied. In the following we shortly discuss a test pattern template for the definition of test patterns, aspects of test patterns mining and of the application of test patterns.

3.1 Test Pattern Template

To unify the pattern definition process and to avoid misunderstandings between developers discussing patterns, a template providing a guideline is needed. Taking the test pattern template provided by [7] as basis, we propose a more refined test pattern template adapted to the TTCN-3 testing domain. Further analysis will be made, to assess how appropriate that template would also be for other test approaches based on formal notation.

3.2 Test Pattern Mining

Test pattern mining is the process of abstracting from existing problem-solution-benefit triples in the test developing process, to obtain patterns suitable for reuse in future contexts. Although the process of going through existing test artefacts and trying to identify patterns for later reuse might appear costly and unrewarding at the first sight, we believe in long term it could effectively help in shortening the test development lifecycle and hence reduce costs.

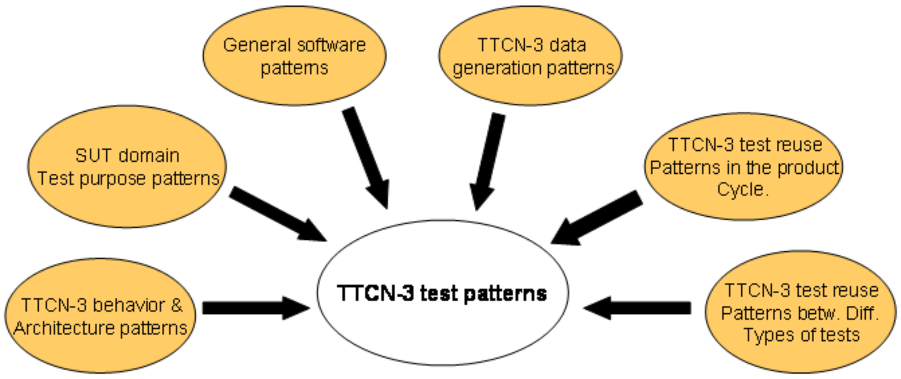
General software patterns
TTCN-3 data generation patterns
SUT domain Test purpose patterns
TTCN-3 test reuse Patterns in the product Cycle.
TTCN-3 test patterns
TTCN-3 behavior & Architecture patterns
TTCN-3 test reuse Patterns betw. Diff. Types of tests

Fig. 3. Elements of Test Patterns can be provided from several activities in and around the Test Developing Process

3.3 Test Pattern Reuse

Reuse of test patterns should basically be independent of the implementation language used for specifying the ATS or for implementing the adaptation to the system and platform environment. However, in a TTCN-3 based test development process the definition of test patterns in TTCN-3 (and extensions thereof) provides several benefits. TTCN-3 test patterns

- Are expressed formally
- Provide means for patterns in the three phase of test system development and for the different approaches of test reuse
- Are defined already in the language of the target test system specification

TTCN-3 provides some concepts for test patterns such as the import mechanism, value parameterization and modifiable templates. Object-based concepts providing further means for the specification and application of test patterns do not exist, but are currently discussed as whether and how they could be included into TTCN-3. Then, better test pattern definitions would be possible. In the meantime we are using specific annotations to TTCN-3 so as to differentiate the generic parts and specific parts of a test pattern. The generic parts are annotated with <> and are to be replaced when applying the test pattern. The specific parts are not annotated. They constitute the essence of that test pattern and should remain untouched when applying that pattern). These annotations are used in the following example.

4 An Example TTCN-3 Test Pattern

To illustrate the ideas presented in the sections above, this section presents an example of a TTCN-3 test pattern. It is a behavioral test pattern for a watch dog on SUT responses.

Name	Timer on transmission
Class	Behaviour
Testing phases	Specification
Testing goals	All
Application domain	Any
Intend	Avoid deadlock situation when transmitting data from test component
Context	For the testing of reactive systems tested typically via interfaces
Parameter	Timer duration
Roles	test component, source port, destination port
Detailed description	After calling a method from a test component or sending a message on a given port, a timer should be started to avoid deadlock in case the SUT does not reply to the function call or to the transmitted message

Example	`function timedSend (template <outMessageType> <outMsg>, <OutPortType> <outPort>, timer <t>, template <inMessageType> <inMsg>, <InPortType> <inPort>)`
	`returns verdicttype`
	`{`
	`        <outPort>.send (<outMsg>); <t>.start;`
	`alt {`
	`       []<inPort>.receive (<inMsg>) {`
	`          <t>.stop; return pass;}`
	`       []<t>.timeout{return fail;}}}`

Consequences	None
Related patterns	Default pattern
Known uses	Protocol testing

5 Conclusions

This paper discusses that developing reusable tests is similar to developing reusable software components and therefore the same techniques and methods can be applied. Requirements for reusable tests were presented on a general level based on past studies as well as new requirements were identified and illustrated using TTCN-3 language.

In addition to the three dimensions of software reuse (granularity, scope and target) presented in [6], test reuse addresses three approaches for reuse. Additionally to domain analysis traditionally required in software reuse, vertical, horizontal or historical reuse is to be considered. Vertical reuse should be considered when the test reuse is applied on an individual product or software component and the interfaces are likely to remain intact. However, if this product is a part of product family or is likely to be a basis for future updates (interfaces are likely to change), horizontal and historical reuse will provide bigger savings in the long run.

The discussed aspects of test patterns can be used as guidelines when designing and implementing reusable tests. Applying these guidelines into practice demands careful consideration to identify the best possible way to utilize and combine them. However, one should notice that striving for the best reusability by combining as many guidelines as possible, will not always produce the best results in terms of usefulness and functionality. Finding the balance between reusability and usability requires consideration. Nevertheless, one of the major strengths of test reuse is that it provides high quality and maintainable tests that are sometimes hard to come up with otherwise. Therefore, test reusability should be an obvious issue when developing tests.

In the ongoing work we will continue to develop test patterns and to apply/propose language extensions for TTCN-3 in order to enable the specification of test pattern. Tool support for these language extensions is also considered.

References

[1] Ch. Alexander: The Timeless Way of Building, Oxford University Press 1979.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: Pattern-Oriented Software Architecture, A System of Patterns, Volume 1, Wiley Series in Software Design Patterns, ISBN 0 471 95869 7, 1996-2001.

[3] I. Jacobson, M. Griss, P. Johnsson: Software Reuse - Architecture, Process and Organisation for Business Success. New York, USA: Addison-Wesley, 1997.

[4] ETSI Draft Report: Methods for Testing and Specification (MTS) M. Frey et al,: Pattern for Test Development (PTD), March 2004.

[5] I. Schieferdecker, T. Vassilou-Gioles:. Tool Supported Test Framworks in TTCN-3. Electronic Notes in Theoretical Computer Science 80. 10 p. http://www.elsevier.nl/locate/entcs/volume80.html, 2003

[6] E.-A. Karlsson:. Software Reuse. A Holistic Approach. New York, NY. John Wiley & Sons. 1995.

[7] Robert V. Binder, Testing Object Oriented Systems: Models, Patterns and Tools, Addison Wesley, 1999

[8] ETSI ES 201 873 01, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part1: TTCN-3 Core Language; ETSI Standard, Feb. 2003

[9] ETSI ES 201 873 05, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part5: TTCN-3 Runtime Interface (TRI); ETSI Standard, Feb. 2003

[10] ETSI ES 201 873 06, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part6: TTCN-3 Control Interfaces; ETSI Standard, Feb. 2003

[11] M. Clifton, Advanced Unit Tests; Part V; Unit Test Patterns; An introduction to the Concept of Unit Test Patterns http://www.codeproject.com/gen/design/autp5.asp, March 2004

[12] John Mc. Gregor, A Parallel Architecture for Component Testing of Object Oriented Software; http://www.cs.clemson.edu/%7Ejohnmc/new/pact/qwpaper.html ,

[13] D. Delano, L. Rising, System Test Pattern Language, AG Communication Systems http://www.agcs.com/supportv2/techpapers/patterns/papers/systestp.htm#TestersMoreIm portant