The Commuting V-Diagram
On the Relation of Refinement and Testing

Bernhard K. Aichernig

United Nations University
International Institute for Software Technology (UNU/IIST)
P.O. Box 3058, Macau
bka@iist.unu.edu

Abstract. This article discusses the relations between the step-wise development through refinement and the design of test-cases. It turns out that a commuting diagram inspired by the V-process model is able to clarify the issues involved. This V-diagram defines the dependencies of specifications, implementations and test-cases in the category of contracts. The objects in this category are contracts defined in the formalism of the refinement calculus. The maps are the refinement steps between these objects. Our framework is able to define the correctness notion of test-cases, testing strategies as refinement rules, and which test-cases should be added under refinement.

Keywords: formal methods, specification-based testing, refinement, refinement calculus, contracts.

1 Introduction

The synergy of formal methods and testing has become a popular area of research. In the last years, test-generation tools have been invented for almost every popular specification language. One reason is the industry’s demand to cut the efforts of software testing. Another is the academics’ insight that testing is complementary to proving the correctness of a program (see e.g. Hoare’s comments on testing in [5]).

However, only little research has been put into the question how the related development techniques such as refinement contribute to testing. Our current research addresses this open issue. In our previous work we have demonstrated that test design can be viewed as a reverse program synthesis problem of finding adequate abstractions [1]. The consequence of this insight is that we are able to define test-case synthesis rules in order to calculate correct test-cases from specifications. The mathematical framework in our work is the refinement calculus of Back and von Wright [6] including a simple but powerful contract language.

Our general approach is able to cover rather different test-selection techniques like domain partitioning [3], interactive scenario selection [2], and mutation testing [4]. These previous work demonstrated that a test-case selection strategy can be represented by means of formal synthesis rules that define how specifications
should be changed into test-cases. This approach is inspired by the refinement calculus where refinement rules define correctness preserving development steps. The difference is that our rules represent the derivation of test-cases by means of abstraction steps.

In this paper we focus on the question: “Which new test-cases are needed if we refine a specification or implementation?”. In order to give a scientific answer, the role of testing and step-wise development has to be clarified. This can be done using a simple diagram (Figure 1) to which we give a precise mathematical semantics.

![The V-Diagram](image)

**Fig. 1.** The V-Diagram.

The V-diagram is inspired by the V process model, a derivative of the waterfall model where the development phases are explicitly linked to testing. Both visualizations stress the importance of testing. The left-hand side of the V in Figure 1 represents the step-wise development of a specification $C_1$ into an implementation $C_n$. In the following $C_1$ to $C_n$ are called contracts representing commitments on different levels of abstractions. All the arrows in the diagram denote refinement. On the right-hand side of the V, the corresponding test-cases are shown. The test-cases are refined in accordance with the contracts. In our view of test-cases, contract $C_i$ must be a refinement of its test-cases $T_i$. Consequently, test-cases can be viewed as a special form of specification. During discussions we have found that not too many colleagues are aware of this fact.

The formal refinement relation between test-cases $T_i$ and a contract $C_i$ can be interpreted in two directions:

- **test-cases as specifications**: if test-cases are given, an implementation or formal specification must be a correct refinement of the intended test-cases.
- **test-synthesis as an abstraction problem**: As a refinement calculus is a technique to derive correct implementations from a specification by following correctness preserving refinement rules, dual abstraction rules can be used to calculate test-cases. The names of the refinement arrows $t_i^{-1}$ in Figure 1 should indicate this reverse process of abstraction.