

1. Introduction

1.1 Requirements Engineering

The module title should be something like “Specification of Software systems” or “Requirements Specification” rather than its current title which is somewhat unsatisfactory.

There are various aspects of “Requirements engineering”, including

- Requirements Elicitation
- Requirements Specification & Documentation

We won’t be concerned with “Requirements Elicitation” in this module but it is worthwhile to give a flavour of what is involved. Thus, there are various techniques used for “Requirements Elicitation”, including

Technique	Major Info Source		Strong on	
	Domain	User	Current	Future
Interview		X	X	
Delphi Technique		X	X	
Brainstorming		X		X
Task Analysis		X	X	
Scenario (Use Case) Analysis		X		X
Ethnography	X		X	
Form Analysis	X		X	
Analysis of natural language descriptions	X		X	
Synthesis of reqts. from existing system	X		X	
Domain Analysis	X		X	
Use of Reference Models	X		X	
Business Process Redesign (BPR)	X		X	X
Prototyping		X		X

Figure 1-1: A sample of requirement elicitation techniques (H. Van Vliet, Software Engineering, Wiley, 2004)

1.2 System Specification Techniques

This module is mainly concerned with system specification techniques for which several techniques are used though we will focus on a limited number. Examples of requirement specification techniques include

- Entity-Relationship Modeling
- Finite State Machines (with state diagrams or statecharts)
- Structured analysis & Design Technique (SADT)

Also, one can apply an object-oriented approach to analysis, supported by UML. In fact, there are a number of OOAD approaches including OMT and FUSION. We will cover some of these in the module.

Specification techniques can be categorised as “formal” or not. Here “formal” means “in some machine processable form”, thanks to a well-defined language syntax and semantics.

Some techniques are “semi-formal” in the sense that they are a combination of machine-processable material (often graphical) and natural language.

1.3 Formal methods' categorisation

There are a number of formal specification techniques that serve to specify the behaviour of abstract data types.

Recall the definition,

“An abstract data type consists of a set of values of some type together with a set of functions that can be applied to these values.”

The techniques can be categorised as one of

- Algebraic specifications
- Model-oriented specifications.

Ex: Specification of a stack of integers: Comparison of Algebraic & Model-oriented

(a) Algebraic Specification

type Intstack;

functions

Create:		→ Intstack
Push:	Intstack X Int	→ Intstack
Pop:	Intstack	→ Intstack
Top:	Intstack	→ Intstack
Iempty:	Intstack	→ Boolean

axioms

Iempty(Create) = true
 Iempty(Push(s,i)) = false
 Pop(Create) = Create
 Pop(Push(s,i)) = s
 Top(Create) = 0
 Top(Push(s,i)) = i

end Intstack;

Notice that the semantics of the functions is given implicitly, through relations between the functions. These relations are called axioms or re-write rules.

Note: The general idea is to group the system's operations by the concept to which they refer, and to define the effect of their combined application. Specification languages such as OBJ, ASL and LARCH rely on this paradigm.

(b) Model-Oriented Specification

let stack = $\langle \dots x_i \dots \rangle$ where x_i is int;

invariant $0 \leq \text{length}(\text{stack})$;

initially stack = nullseq;

function

 push(s: stack, x:int)

pre $0 \leq \text{length}(s)$

post $s = s' \sim x$,

 pop(s: stack)

pre $0 < \text{length}(s)$

post $s = \text{leader}(s')$,

 top(s: stack) **returns** x:int

pre $0 < \text{length}(s)$

post $x = \text{last}(s')$,

 isempty(s: stack) **returns** b:boolean

post $b = (s = \text{nullseq})$

Note: Some abstract object is chosen to represent the data type to be specified. The semantics of the various functions is given explicitly by specifying their effect on the abstract object. In the example, the abstract object is the sequence $\langle \dots x_i \dots \rangle$. The most well known model-oriented specification languages are Z and VDM. A large part of this module is focussed on Z.

Note: Many specification techniques, including Z, are focussed on specifying the functionality of systems. However, it is also necessary to specify non-functional aspects such as performance, safety or reliability.

1.4 Perspective on specification & formalization

It should be borne in mind that specifications may be used for a number of different purposes:

- They may clarify issues
- They may serve as a contract
- They may serve to aid the design process
- They may be the starting point for implementation
- They may be the starting point for validation & verification

Each of the above is best served by having a very precise specification, which is where formalization can be very important.

It is commonly held that a “specification” defines what should be done rather than how to do it. This is reasonable but a bit of a simplification as the border line between what and how may not be clear-cut.

Some people would say that a program is a “specification”. Also, one may sometimes have “executable” specifications (which need not be the final program).

A useful view of things is to regard a specification as a “program” (not necessarily executable) at an abstract level. One then proceeds by adding more and more concrete details until finally the end product, the actual program, is reached. The actual program is of course formal in that it is completely machine-processable (assuming that it is in a well-defined programming language).

There are various strategies by which formalization can be introduced into the transformation process from requirements to final program. Thus, for example, we may have

- [Extreme but common] Postpone formalization until the very end. All documentation, initial requirements to detailed design, is informal (e.g. in a natural language with various graphical techniques).
- One or more intermediate representations may help to move from the initial informal specification to the formal specification, using semi-formal techniques as “mediators”.

1.5 *Whats’s next?*

In the next section (§2 Z language – Iteration 1), we present the Z language through a series of progressively more detailed examples. We will introduce the elements of Z bit by bit rather than by a complete definition at the outset.

In the following section (§3 Z language – Iteration 2), we will present more example specifications, and will provide a systematic review of the Z language by reference to Spivey’s reference manual of Z notation.

Later sections will deal with other topics.