

SOFTWARE FUNDAMENTALS: COLLECTED PAPERS - CH. 3,6,7

Tamim Sookoor

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22904
`sookoor@cs.virginia.edu`

June 14, 2007

MOTIVATION

- Standard logic requires total functions

EXAMPLE

$$\sqrt{x}$$
$$\sqrt{|x|}$$

$$((x > 0) \wedge (y = \sqrt{x})) \vee ((x \leq 0) \wedge (y = \sqrt{-x}))$$

- Advanced logic deals with partial functions in complex ways

DEFINITION

A *total function* is a function defined on a domain that includes all possible values of their arguments.

A *partial function* is a function whose value has not been defined for certain values of the argument.

- Enable mathematics to be used for precise documentation



PREDICATE LOGIC

- Allow partial functions
- Eliminate “undefined” as a truth value
- Restrict the set of primitive predicates
- Eliminate distinguished member (*)



BASIC DEFINITIONS

- *Simple tuple*: An ordered set of one or more members of a finite set of values, U
- *Simple n -tuple*: An ordered list of n members of U
- *Tuple*: An ordered list of one or more simple tuples
- *n -Tuple*: A tuple containing n elements, each of which is a simple tuple
- *Relation*: A set of tuples
- *Binary relation*: A set consisting entirely of pairs
- *Function*: A one-to-one binary relation
- *Predicate*: A function whose range contains only *true* and *false*
- *Characteristic predicate*: A predicate, for a set of simple tuples X , whose domain is S and whose value, for a simple tuple b , is *true* if and only if b is a member of X .



SYNTAX OF LOGICAL EXPRESSIONS

- f_1, \dots, f_k are names of functions
- R_1, \dots, R_m are names of characteristic predicates
- *Function application*: A string of the form $f_j(V)$ where V is a comma separated list of items
- *Primitive expression*: A string of the form $R_j(V)$

EXAMPLE

divisible_by(x,5)

- *Predicate expressions*: All primitive expressions

EXAMPLE

$(x, P) - (P)$



MOTIVATION

- Software is difficult to understand, change, and maintain
- Suggested solutions:
 - Modularity and information hiding
 - Formal specifications
 - Abstract interfaces
 - Cooperating sequential processes
 - Process synchronization routines
 - Resource monitors
- These techniques are not being used because:
 - their usefulness has not been proven for programs with stringent resource limitations
 - some of them do not have fully worked out examples.
- The paper presents a new technique for making requirements specifications precise, concise, unambiguous, and easy to check for completeness and consistency.



OBJECTIVES OF REQUIREMENTS DOCUMENT

- Specify external behavior only
- Specify constraints on the implementation
- Be easy to change
- Serve as a reference tool
- Record forethought about the life cycle of the system
- Characterize acceptable responses to undesired events



REQUIREMENTS DOCUMENT DESIGN PRINCIPLES

- State questions before trying to answer them
- Separate concerns
- Be as formal as possible



DEFINITIONS

- *Data item*: A unit for each input or output that changes value
- *Software*: A set of functions associated with output data items
- *Demand function*: A function requested by the occurrence of some event every time it is performed
- *Periodic function*: A function performed repeatedly without being requested each time



TEXT MACRO

- Terms used to keep function descriptions concise
- Defines quantities that affect output values, but that cannot be directly obtained from inputs
- Serves as abbreviations for compound conditions that are frequently used or very detailed



MODES

- Used to organize conditions into groups to keep the function descriptions simple
- Given a short mnemonic name enclosed in asterisks
- Current mode defined by history of events that have occurred in the program

EXAMPLE

```
*DIG* TO *DI*  
@T(!latitude > 70)  
@(/IMSMODE/= $Iner$)WHEN(!Doppler coupled!)
```



MOTIVATION

- Programs with a large amount of functionality are complex
- Criteria for decomposing programs necessary



CONTRIBUTIONS OF THE PAPER

- A module is a work assignment
- Every information hiding module should be accessible only through provided interfaces
- Modularization should occur at design time
- Modules should be evaluated by asking what changes it accomodates
- Hierarchical structuring and modular decomposition are two different concepts



BENEFITS OF MODULARIZATION

- Manager: Shortened development time
- Marketer: Product flexibility
- Programmer: Comprehensibility



CRITERIA FOR MODULE DECOMPOSITION

- Data structures
- Routines and call sequences
- Formats of control blocks
- Character codes, alphabetic orderings, and similar data
- Sequence in which an item will be processed

