Learning Objective

To give an appreciation of the strengths and limitations of FMs as an important part of the *Software Designers Repertoire*. *Formal descriptions* can provide a powerful aid to developing a design, especially when issues such as *consistency and verification* are considered.

Frederick T Sheldon
Assistant Professor of Computer Science
University of Colorado at Colorado Springs
Agenda

- The case for rigor
- Model-based strategies
  - Overview
  - VDM / VDM Process / VDM Heuristics
- Property-based strategies
  - Overview
  - Algebraic Specification: representation part
  - Algebraic Specification: process part
  - Heuristics for property-based specification
Problem

⃝ Methods and tools are needed for software specification and design that have mathematical underpinning… not just systematic

⃝ Formal methods have
  ⋐ Fairly simple process parts
  ⋐ Relatively few established design heuristics
  ⋐ But (in comparison) very powerful representation parts

⃝ Often termed Formal Description Techniques
  *FDTs*
Motivation
Why Is This An Important Problem?

- Is the "Software" doing what it is supposed to do?
- Is the system doing what it is supposed to do?

**System Development**
- What the "System" is supposed to do.
- System Definition
  - Requirements Definition
    - Software Requirements Generation
      - Software Design
        - Coding & Component Testing
        - Integration and System Testing
      - System Validation
        - Software Validation Testing
        - System Validation Testing
        - Hardware Software Integration

**Software Development**
- What the software is supposed to do.
- Software Design
  - Software Requirements Generation
    - System Definition
Motivation — The Case for Rigor

⚠ Systematic systems for *specification and design* lack a firm syntax and well-defined semantics

⚠ Need for the application of mathematical techniques in reasoning about a design and its properties

⊕ Verification:
  • Seeking to remove ambiguity
  • Enforce a greater attention to detail
  • Verify the fidelity between design transformations and implementation
Roles of Formal and Systematic Description Techniques in the SLC

⊗ SDTs are better for:
  ⊕ Reqvs analysis (including user interactions)
  ⊕ System design (architectural decisions)

⊗ FDTs
  ⊕ Specify system properties during requirements spec.
  ⊕ Specify the detailed form a solution in the detailed design

⊗ Fig. 14.1
FDTs Uptake in Industry Limited

- Conservatism of most project managers
- Need for familiarity with logic / discrete math
- Existing forms not universally applicable or not suited for all problems
- Limited tool support
- Overselling → Unreasonably high expectations → Disillusionment
Use of Formal Methods

⊗ These methods are unlikely to be widely used in the foreseeable future. Nor are they likely to be cost-effective for most classes of system

⊗ The will become the normal approach to the development of safety critical systems and standards

⊗ This changes the expenditure profile through the software life cycle
Expenditure Profile Changes

Without Formal Specification

Validation
Design and Implementation
Specification

With Formal Specification

Validation
Design and Implementation
Specification

Cost

Maintenance
Formal Spec Langs Provide…

- Notation – the *syntactic* domain
- Universe of objects – the *semantic* domain
- Rules for stating which objects satisfy each specification
- FDTs can be grouped into 2 categories:
  - Model-based (e.g., VDM and Z [or Zed])
  - Property-based (e.g., Axiomatic or Algebraic forms)
General Categorization of FMs

 ※ Model-based
  ⊕ Use structures such as
      sets, functions, tuples and
      sequences

※ Property-Based
  ⊕ Axiomatic forms use
      procedural abstractions
      based on 1st order logic
  ⊕ Algebraic forms model
      data abstractions (axioms
      in the form of equations)

※ Fig. 14.2
Two Further Classifications

⊗ Visual languages
  ⊕ Graphic forms provide the syntactic content
  ⊕ Examples include Statecharts and Petri nets

⊗ Executable forms
  ⊕ Via an interpreter (e.g., Prolog, and PAISley)
FM... the Jury is Still Out

- FMs combine very strong representation parts with weak process parts
  - Process involves stepwise refinement
- The roles and uses of design heuristics are harder to identify
- Are FMs essentially domain specific
  - Some problems are more readily solved via FMs and others are not!
Model-based Strategies

⊗ Reification is shown here... ⊗ Fig. 14.3
Characteristics of Model-based FDTs

Use a mathematical form to construct a model of the system...

...to reason about properties and behavior.

While the property-based forms focus on describing the external features of the system,...

...the Model-based approach focuses on the mechanisms used to produce those features!
VDM: Model-based approach

Thompson emphasized mathematical rigor:

- In preference to complete formality!
- Intuition often used to provide correctness arguments,
- Full verification applied sparingly when absolutely necessary

VDM promotes the use of reification through a series (sequence) of models:

- Abstract to concrete through an explicit model of the state of the system
VDM representation

Two major components

+ Definition of *abstract variables* used to describe the internal state of the model

+ Definitions of the operations and functions that act on the variables making up the model
  - Operations that may be available externally

+ Similar to traditional imperative programming languages (e.g., Modula-2, Ada)
VDM: Typographical Conventions

- User defined *OPERATIONS* are printed in upper-case serif italics
- Identifiers of *types* are printed in serif italics
- Identifiers of variables are printed in serif roman type in declarations
- Identifiers of *keywords* are printed in bold sanserif type
- Type identifiers begin with an upper-case letter followed by a sequence of lower-case letters
- The constants of scalar types are named using upper-case serif italic letters only
- Extensions such as -set or -list are printed in a sanserif typeface
Data Forms

⊗ Simple types. . .
   ⊕ Built-ins (e.g., Int(), Nat())
   ⊕ Sets
   ⊕ Lists (sequence, tuple)

⊗ Complex types. . .
   ⊕ Records
   ⊕ Mapping (special form of function that maps between sets)
Operations and Functions
Standard Operators of Predicate Logic

⊗ ~ not
⊗ ∧ and
⊗ ∨ or
⊗ ≡ is equivalent to (iff)
⊗ ∀ for all (universal quantifier)
⊗ ∃ there exists (existential quantifier)
⊗ ∃! there exists exactly one
⊗ Let clause … allows an expression to be named
Example VDM Specification

Figure 14.4
Defining the System Operations

Three parts (not all are required)

- `ext` – parts of the state accessed in the operation (rd/wr)
- `pre` – precondition forming a predicate condition under which the operations are defined
- `post` – showing how values of variables are modified

Also:

- Invariants – predicates which define additional constraints on the values that variables may assume
- Comments – improved readability
VDM Process

Figure 14.5

⊗ Reification and verification using proofs
   ⊕ Repeatedly adding more detail to a specification in terms of...
      • Data structures
      • Operations (preformed on the data structures)
   ⊕ Until an implementation level specification has been obtained
   ⊕ Scope of choice is limited at each step which may be considered to ensure consistency
Property-based Strategies

⊗ Algebraic specification technique
   ⊕ An object class or type is specified in terms of the relationship(s) between the operations defined on that type
   ⊕ Representation part:
      • Introduction
      • Informal description
      • Signature
      • Axioms
Algebraic Specification

Introduction

Importing

A sort and its operations brings them into the scope of the new specification.

Enrichment

Allows a new sort to be defined that inherits the operations and axioms of another specification.

Similar to the inheritance mechanism used in OO.
Algebraic Specification

Informal Description and Signature

⊗ Informal description
  ⊕ Textual comments used to explain the mathematical formalism

⊗ Signature
  ⊕ Define the external appearance of an object by describing its basic properties using a set of operations
    • Constructor ops (create, update, add)
    • Inspection ops (used to evaluate the attributes of the entity’s sort)
Algebraic Specification

Axioms

What is an axiom

- An established rule, principle or law
- Defines the inspection operations in terms of the constructor operations
- The main technical problem of developing algebraic specifications

Thus, a set of mathematical expressions are developed that define the relationships of operations in the signature

- Constructors and Inspectors
Algebraic Specification

Process and Heuristics

⊗ Most literature is concerned with describing the form of a specification that its derivation
⊗ Techniques for ensuring completeness and correctness are well established
⊗ The algebra of these specifications bear a familiar form
⊗ Side effect: generating axioms effectively generates guidelines for testing the implementation . . . yes!
Summary

Formal descriptions can provide a powerful aid to developing a design

- Consistency
- Verification

Design techniques needed for the derivation of a Formal Specification are much less well developed

- …. as opposed to mathematical techniques
The Evolution of SW Design Practices

Experiences from the past

Design assessment criterion

- Efficiency of operation, memory use or secondary storage

Current assessment criterion include:

- Modularity, reuse, separation of concerns
- Information hiding and conceptual integrity
Current SW Design Practices I

- Identify the right set of abstractions and their relationships
- Capture designers experience as a set of rules or rules of thumb (heuristics)
- How to determine when a good solution has been identified
- No panacea in terms of a method for all problems
Current SW Design Practices II

 Criterion for a good design solution

 - Change in the problem description would require minimal change in the design abstraction

 Trends in terms of design abstractions:

 - Use of increasing number of viewpoints
 - Specialized adaptations of traditional methods toward object oriented forms that require a balance between . . .
  - Function, behavior, structure and data-modeling
Trends in SW Design Abstractions

- Increasing degree of complexity in design procedures
- CASE tools encompassing a wide range of support forms
  - Upper-CASE
  - Lower-CASE
- CASE tools bind the user to themselves
Trends in SW Design Abstractions
(continued)

知名企业 may actually constitute a barrier to communication in design … which is fundamentally a group activity

知名企业 Promotes the NEAT_DIAGRAM syndrome which can inhibit change (and possibly drastic refinement)
Future Developments

The complexity of software is an essential property of software (non accidental). Hence, descriptive forms that abstract away its complexity often abstract away its essence [Brooks, F.]

SWD Methods should look to more powerful paradigms, instead of simpler ones. Therefore,

- Need to encapsulate design expertise (like reuse)
- Tools that are intelligent (I.e., domain specific, and embody semantic knowledge to help assess the consequences of their decisions (trade-offs)

Move away from procedural forms as the defacto software design approach