CSE503: Software Engineering

David Notkin
University of Washington
Department of Computer Science & Engineering
Winter 2002

C.A.R. Hoare, 1988

Of course, there is no fool-proof methodology or magic formula that will ensure a good, efficient, or even feasible design. For that, the designer needs experience, insight, flair, judgment, invention. Formal methods can only stimulate, guide, and discipline our human inspiration, clarify design alternatives, assist in exploring their consequences, formalize and communicate design decisions, and help to ensure that they are correctly carried out.

Model-oriented specifications

- Model a system by describing its state together with operations over that state
  - An operation is a function that maps a value of the state together with values of parameters to the operation onto a new state value
- A model oriented language typically describes mathematical objects (e.g. data structures or functions) that are structurally similar to the required computer software

Z ("zed")

- Probably the most widely known and used model-based specification language
- Good for describing state-based abstract descriptions roughly in the abstract data type style
- Based on typed set theory and predicate logic
- A few commercial successes
  - I'll come back to one reengineering story afterwards

The basic idea

- Static schemas
  - States a system can occupy
  - Invariants that must be maintained in every system state
- Dynamic schemas
  - Operations that are permitted
  - Relationship between inputs and outputs of those operations
  - Changes from state to state

Illustrative example (Zeil)

- I'll sketch out a standard Z-style example
- Z relies heavily on non-standard characters and formatting, which I will only approximate
  - The reading includes a similar example
  - And uses the Z notation

Phone directory: static schema

- A static schema has three parts
  - A name
  - A set of declarations that define the state
  - A set of invariants that constrain all legal states
- `PhoneDB`
  - `members: P Person`
    `telephones: Person <-> Phone`
  - `dom telephones SUBSET-OF members`

7

Type of `members: P Person`

- Atomic elements, like `Person` and `Phone`, represent sets of values
- `P Person` represents the power set of `Person`, the set of all sets taken from `Person`
- So, `members` is one of those: a set of `Person`

8

Type of `telephones: Person <-> Phone`

- `telephones` is a relation between `Person` and `Phone`
- That is, it is a set of pairs, where the first element is taken from `Person` and the second is taken from `Phone`

9

Invariant

`dom telephones subset-of members`

- This is an invariant that defines a constraint on all legal states of `PhoneDB`
- The domain (the set of first elements in the pairs) of `telephones` must only contain elements that are in `members`
- Without this invariant, there would be no restrictions nor relationship between `members` and `telephones`
- When we define operations that can modify the state of `PhoneDB`, they are obligated to maintain (prove) that this invariant is maintained

10

Example: a legal `PhoneDB` state

- `Person: { jerre, hellmut, bob, paul, jean-loup, ed, david}`
- `Phone: { 5-3798,3-2121,3-5010,3-4755,5-1376, 3-1695,3-2969,3-6175,6-4368}`
- `members: { jerre, hellmut, jean-loup, ed, david }`
- `telephones: { (jerre |->3-6175), (hellmut |-> 3-6175), (jean-loup |-> 5-1376), (ed |-> 3-4755), (david |-> 5-3798) }`
- `|->` is a "maplet", essentially a pair

11

A few notes on the example

- The elements of `Person` and `Phone` are atomic: they have no required syntax nor semantics
- `telephones` is a relation, not a function; so adding the tuple `(david |-> 3-1695)` to it is perfectly legal
- And it already contains two tuples with the same range (second element of the pair): `jerre` and `hellmut` share `3-6175`
- Z, of course, has and uses functions (both partial and total)
  - But they are notational conveniences, since one can write invariant that constrain relations to be functions

12

Example: an illegal PhoneDB state

- Person: { jerre, hellmut, bob, paul, jean-loup, ed, david, jonathan }
- Phone: { 5-3798,3-2121,3-5010,3-4755,5-1376, 3-1695,3-2969,3-6175,6-4368,1-2345}
- members: { jerre, hellmut, jean-loup, ed, david }
- telephones: { (jerre |->3-6175), (hellmut |-> 3-1675), (jean-loup |-> 5-1376), (ed |-> 3-4755), (david |-> 5-3798), (jonathan |-> 1-2345) }
- This would be perfectly legal in the absence of the invariant: but jonathan, while being an element of Person, is not an element of members

Dynamic schema: specifying state transitions

- Static schema specify legal states
- But we also need to specify operations that transform one legal state into another legal state
- Dynamic schema have (just like static schemas)
  - A name
  - A set of declarations
  - A set of invariants that relate the set of declarations to one another
- However, the declarations used are richer

Example declaration

- Declaration: DELTA PhoneDB
- A DELTA declaration introduces pre- and post-states for each of the declarations in the named schema
  - members and members'
  - telephones and telephones'
- The unprimed names represent the pre-states and the primed names the post-state
- Any invariants must hold on the pre-state and then again on the post-state
  - dom telephones SUBSET-OF members
  - dom telephones' SUBSET-OF members'

Example dynamic schema

- Name: AddEntry
- Declarations:
  - DELTA PhoneDB
    name? : Person
    newnumber?: Phone
- Invariants
  - name? IS-ELEM members
    (name? |-> newnumber?) NOT-ELEM telephones
    telephones' = telephones UNION (name? |-> newnumber?)
    members' = members
- This may or may not be what you expect from AddEntry, but it is clear about key issues: for instance, it only adds new phone numbers for existing members

What if…

- telephones' = telephones UNION (name? |-> newnumber?)

  was replaced with
- (name? |-> newnumber?)IS-ELEM telephones'

Returning information

- GetNumber
- XI PhoneDB
  name?: Person
  number!: **P** Phone
- name? IS-ELEM members
  number! = { n : Phone | ((name? |-> n) IS-ELEM telephones)}
- The XI declaration is the equivalent of DELTA along with the following invariants that guarantee no change to the PhoneDB declarations
  - members = members'
  - telephones = telephones'

Error conditions

- Note that the dynamic schema we've seen so far just specify what happens in the "good cases"
  - Nothing is specified for the error conditions
  - What happens with `AddEntry(mork,0-1010)`?

19

Specify in separate schema

- `NotMember`
- `XI PhoneDB`
  `name? : Person`
  `report! : Report`
- `name? NOT-ELEM members`
  `report! = 'not a member'`

20

But it's still entirely separate

- `Success`
- `report! : Report`
- `report! = 'OK'`
- And then the coolest thing in Z…(at least notationally) is the schema calculus
- `AddEntryWithError ==`
  `(AddEntry AND Success) OR`
  `NotMember`
- This is the same as a dynamic schema in which the three schema are commingled according to the stated logic
  - They are "pinned" together by shared names

21

Z/CICS

- Z was used to help develop the next release of IBM's CICS/ESA_V3.1, a transaction processing system
  - Integrated into IBM's existing and well-established development process
  - Many measurements of the process indicated that they were able to reduce their costs for the development by almost five and a half million dollars
  - Early results from customers also indicated significantly fewer problems, and those that have been detected are less severe than would be expected otherwise

22

1992 Queen's Award for Technological Achievement

- "Her Majesty the Queen has been graciously pleased to approve the Prime Minister's recommendation that The Queen's Award for Technological Achievement should be conferred this year upon Oxford University Computing Laboratory.
- "Oxford University Computing Laboratory gains the Award jointly with IBM United Kingdom Laboratories Limited for the development of a programming method based on elementary set theory and logic known as the Z notation, and its application in the IBM Customer Information Control System (CICS) product. ...
- "The use of Z reduced development costs significantly and improved reliability and quality. Precision is achieved by basing the notation on mathematics, abstraction through data refinement, re-use through modularity and accuracy through the techniques of proof and derivation.
- "CICS is used worldwide by banks, insurance companies, finance houses and airlines etc. who rely on the integrity of the system for their day-to-day business."

23

Pros and cons?

- Your turn…

24