CSE584: Software Engineering

Lecture 6 (May 6, 1997)

David Notkin
Dept. of Computer Science & Engineering
University of Washington
www.cs.washington.edu/homes/notkin

UW CSE
CSE584: Software Engineering

Lecture 6, Outline [approximate minutes]

- Requirement specification terminology [30]
- Informal approaches to requirements specification [5]
- Formal methods overview [20]
- Break [10]
- Basic Z example (telephone) [45]
- Very basic Statecharts example [15]
- Writing a spec (in small groups) [30]

Requirements specification

- Defines the software to be built
- Historically written in natural language
  - Natural language is inherently ambiguous
  - Not always especially concise
- There are a number of approaches to overcome problems with natural language requirements specification definition
  - Much of the next part of lecture is due to M. Jackson

Ambiguity

- "Hard hats must be worn before entering the construction area."
  - At the UW EE/CSE building construction site entrance
- Michael Jackson's favorite example
  - At the foot of an escalator

Shoes Must Be Worn

Dogs Must Be Carried

Shoes and dogs

- Your interpretation goes here

Optative vs. indicative moods

- Jackson observes that specifications often improperly (or confusingly, at least) mix two forms of statements
- The indicative mood states a fact

$$\forall x \bullet \texttt{(OnEscalator(x)} \Rightarrow \exists y \bullet \texttt{(PairOfShoes(y)} \wedge \texttt{IsWearing(x,y))}$$

- The optative mood states a wish
  - What happens if someone not wearing shoes gets on the escalator?

Optative and indicative

- Indicative properties are those that are invariantly true regardless of the program
  - In essence, they describe the operating environment for a program
- Optative properties are those that you want to achieve
  - In essence, these are the requirements

Optative or indicative?

- Specifying the minimal separation distance between airplanes in an air traffic control system
  - If you make it indicative (that is, state it as an invariant) then you can't describe the requirements, which are intended to *ensure* that this separation holds

Jackson examples: which mood?

(a) The lift never goes from the nth to the n+2nd floor without passing the n+1st floor
(b) The lift never passes a floor for which the floor selection light inside the life is illuminated without stooped at that floor
(c) If the motor polarity is set to up and the motor switch setting is changed from off to on, the lift starts to rise without 250 msecs.
(d) If the upwards arrow indicator at a floor is not illuminated when the lift stops at the floor, it will not leave in the upwards direction.
(e) The doors are never open at a floor unless the lift is stationary at that floor.
(f) When the lift arrives at a floor, the lift-present sensor at the floor is set to on.
(g) If an up call button at a floor is pressed when the corresponding light is off, the light comes on and remains on until the call is serviced by the lift stopping at that floor and leaving in the upwards direction

Principle of Uniform Mood

- Indicative properties and optative properties should be entirely separated in a specification document
  - Reduces confusion of both the authors and the readers
- If the software works right, both sets of properties will hold as facts

Defining terms

- A key aspect of ambiguity in requirements documents arises when terms aren't well-defined
  - When is something a "pair of shoes"?
  - When is a pair of shoes "being worn"?
  - What is a "dog"?

"dog" (noun)

- OED has 15 definitions (11K words in full definition)
- Webster's 11 definitions include
  - » a highly variable domestic mammal (*Canis familiaris*) closely related to the common wolf
  - » a worthless person
  - » any of various usu. simple mechanical devices for holding, gripping, or fastening that consist of a spike, rod, or bar
  - » FEET
  - » an investment ... not worth its price
  - » an unattractive girl or woman

“shoe” (noun, Webster’s)

- Six definitions including
  - » an outer covering for the human foot usu. made of leather with a thick or stiff sole and an attached heel
  - » another's place, function, or viewpoint
  - » a device that retards, stops, or controls the motion of an object
  - » a device (as a clip or track) on a camera that permits attachment of accessory items
  - » a dealing box designed to hold several decks of playing cards

UW CSE CSE584: Software Engineering Notkin (c) 1997 13

Designation

- A *designation* defines a term through a recognition rule
- Allows one to decide whether a phenomenon satisfies the designation

```
x is a human being     Human(x)
x is male              Male(x)
x is female            Female(x)
x is the genetic
  mother of y          Mother(x,y)
x is the genetic
  father of y          Father(x,y)
```

UW CSE CSE584: Software Engineering Notkin (c) 1997 14

Using designations

- It’s important to give enough definition to make sure all agree on the meaning
  - This takes some leap of faith, especially in domains in which people have little shared experience
- Allows refutable statements to be made about the requirements

  $\forall$ x,y $\bullet$ ((Human(x) $\wedge$ Mother(x,y)) $\Rightarrow$ (Female(x) $\wedge$ Human(y)))

UW CSE CSE584: Software Engineering Notkin (c) 1997 15

Refutable descriptions

- Having refutable descriptions is useful
- The primary reason is that such a description can be used to determine whether a program satisfies a requirement
- Another is that it may help you think more clearly about what you are saying

UW CSE CSE584: Software Engineering Notkin (c) 1997 16

More designations?

- Usually you end up having more terms you need to define
- Should you designate them?
  - x is the genetic brother of y   Brother(x,y)
- An alternative is to define them in terms of existing designations
  - Brother(x,y) $\cong$
    Male(x) $\wedge$ $\exists$ f $\bullet$ (Father(f,x) $\wedge$ Father(f,y) $\wedge$
    $\exists$ m $\bullet$ (Mother(m,x) $\wedge$ Mother(m,y)) $\wedge$ x $\neq$ y

UW CSE CSE584: Software Engineering Notkin (c) 1997 17

Definitions

- Definitions define terms in terms of existing designations
  - They are macros, in essence
- They can simplify what you can talk about
  - But they don’t fundamentally change what you can talk about
- Definitions can’t be right or wrong
  - Just well-formed (or not) and useful (or not)

UW CSE CSE584: Software Engineering Notkin (c) 1997 18

Designations in Z

- The Z specification language defines atomic elements
  - `[Book, Person]`
- This indicates that there are Books and there are Persons
  - Books are always Books; Persons are always Persons
  - Nothing can be both a Book and a Person
- The associated natural language describes the actual designations

UW CSE CSE584: Software Engineering Notkin (c) 1997 19

More on designations

- Writing designations can help clarify your thinking
  - `c is a customer` `Customer(c)`
  - `e is an employee` `Employee(e)`
- Is the time period material?
  - `c is a customer in v` `Customer(c,v)`
  - `e is an employee in v` `Employee(e,v)`
- Can an employee be a customer?

  $\neg\exists$ v,x • (Customer(x,v) $\wedge$ Employee(x,v))

UW CSE CSE584: Software Engineering Notkin (c) 1997 20

Which to use?

- The distinction may also help clarify
  - `m is a member during v` `Member(m,v)`
  - `m enrolls at time e` `Enroll(m,e)`
  - `m resigns at time e` `Resign(m,e)`
- This approach may lead to confusion
  - Must you be a member to enroll?
  - Are you a member after you resign?
- Instead *define* membership in terms of the enrolls and resigns designations

UW CSE CSE584: Software Engineering Notkin (c) 1997 21

Term limits

- None of this is magic
- The distinction in terminology between designation and definition is just another way to help you think more clearly when writing specifications

UW CSE CSE584: Software Engineering Notkin (c) 1997 22

Informal approaches

- Running plain text requirements specifications are increasingly less common
- There are a number of approaches between this and formal specifications that give varying degrees of leverage

UW CSE CSE584: Software Engineering Notkin (c) 1997 23

"Will" and "Shall"

- Some government groups write requirements with specified meanings for "will" and "shall" and "may" and such
  - "shall" is a requirement
  - "may" is an optional requirement
  - "will" describes something not under control of the system
- Not always too clear

UW CSE CSE584: Software Engineering Notkin (c) 1997 24

Structured requirements

- I
  - I.A
    - I.A.ii
      - I.A.ii.3
        - I.A.ii.3.q

Notkin (c) 1997 25

Formal methods

- The original use of formalism in software engineering was for proving the equivalence between a specification and an implementation
  - This had a number of problems
- But there has been a resurgence of interest in formal methods
  - Mostly due to potential usefulness in specification

Notkin (c) 1997 26

Potential benefits

- Increased clarity
- Ability to check specifications for internal consistency
- Ability to prove properties about the specification

- Not always worth the effort

Notkin (c) 1997 27

Styles of formal specifications

- Model-oriented (e.g. Z, VDM)
- Algebraic (e.g. OBJ, Larch)
- Process Model (e.g. CCS, CSP)
- Finite State-based (e.g. Statecharts, RSML)
- Logical, constructive, multi-paradigm, broad spectrum, ...

Notkin (c) 1997 28

Model oriented

- 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

Notkin (c) 1997 29

Algebraic specifications

- Represent structures as algebras
  - Represent results as compositions of operations, not as explicit state
  - Closely related to ADTs
- Algebraic methods tend to provide less implementation bias than some other methods

Notkin (c) 1997 30

Process based specifications

- For describing concurrent systems
- Also algebraic in nature, but focus on processes that can be composed over a variety of operators (such as run in parallel)

Finite state based specifications

- Represent system as a finite state machine
- Transitions fired by external (and maybe internal) events
- Often useful in describing aspects of embedded systems
  - Inputs from sensors, outputs to actuators

Examples we'll look at

- Model based specs
- Algebraic specifications
- Hierarchical state specifications

Telephone features in Z

- Due to Mataga and Zave
  - Information and Software Technology, 1995
- Telephone interface specification
  - How features are invoked by the user
- Connections specification
  - Consequences of interactions such as call processing

Background

- Externally observable behavior of a telephone switch
  - Not the details of switching
- Multiplexing telephones
  - Can handle multiple calls (but talk on only one at a time)
- Features on top of POTS (**P**lain **O**ld **T**elephone **S**ervice)

More background

- Each physical phone can be thought of as having multiple virtual telephones (VTs)
  - Each VT has a "call appearance" with a button and indicator lights
- The VTs on a phone can have different directory numbers (DNs)
  - A single DN may appear on multiple phones
- Simple correspondences with phones, calls and DNs no longer exist

Features

- Bridging
  - Picking up a VT that shares a DN with a VT that is already actively involved in a call
- Hold
  - Suspend call, switch to another VT
- Speed calling
- Conferencing
  - Merge two calls

Features

- Transfer
  - Like conference, but drop the original party
- Drop
  - Drops most recent party on conference call
- Call pickup
  - Answer a call ringing on somebody else's telephone

Features

- Call forwarding All
  - All calls to a DN are forwarded
- Call forwarding Busy
  - Forward a call if no VT for a DN is available
- Call Forwarding Don't Answer
  - Forward if a ringing phone is not answered within a certain amount of time

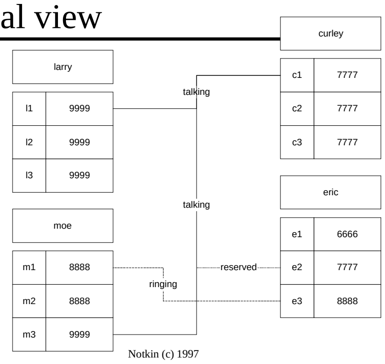
Appropriate level?

- A specification must choose a level that is appropriate
- In this case, a number of issues are not addressed in this specification
  - Translating dialed digits into DNs
  - Any analog or digital processing of sound
  - Etc.

Basic model

- Each telephone has a set of VTs
- Each VT has its own DN
- Each VT has a state that describes its call processing status
  - idle, dialing, ringing, etc.
- A call associates VTs

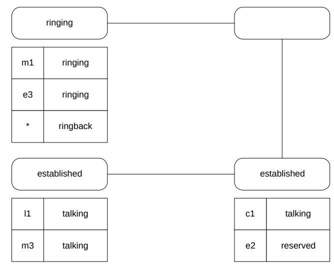
Physical view



Logical alternative

- A logical alternative, at this level of abstraction, is to specify clusters of all VTs with the same DN
  - This can clean up the specification of a call, since it can be represented as a link between clusters (not between VTs or DNs)

Clusters



A scenario

- Larry (VT l1) calls DN 777
- This rings VT c1 and VT e2
  - curley answers at c1
- Later moe joins the call by bridging
  - By picking up m3, which was reserved by the 1st call
- curley on VT c2 calls 8888
  - Ringing m1 and e3 (and a tone generator)
  - While ringing, curley invokes a conference, merging the calls and dropping c2

Empty cluster…and

- It's there to preserve the topology for a later Drop
- The point (in the lecture) isn't the details, but rather the complexity even in this relatively small example

Z schemas

- On the white board
  - [See the paper if you didn't come to class and you want to see them]

Statecharts

- Statecharts is a visual specification language for defining finite state machines
- Perhaps the central feature is that the state description is hierarchical
  - This allows much smaller descriptions for what may be very large state machines