Lecture 11: How Much Formality?

Last Week:
Change and Evolution
Software Evolution
Traceability
Inconsistency

This Week:
How much formality?
Formal Modeling Techniques
Appropriate Uses of FM
Tips on formal modeling

The End!

Formal Methods in RE

➜What to formalize in RE?
- models of requirements knowledge (so we can reason about them)
- specifications of requirements (so we can document them precisely)

Why formalize in RE?
- To remove ambiguity and improve precision
- Provides a basis for verification that the requirements have been met
- Allows us to reason about the requirements
  - Properties of formal requirements models can be checked automatically
  - Can test for consistency, explore the consequences, etc.
- Allows us to animate/execute the requirements
  - Helps with visualization and validation
- Will have to formalize eventually anyway
  - RE is all about bridging from the informal world to a formal machine domain

Why people don't formalize in RE
- Formal Methods tend to be lower level than other analysis techniques
  - They force you to include too much detail
- Formal Methods tend to concentrate on consistent, correct models
  - …but most of the time your models are inconsistent, incorrect, incomplete…
- People get confused about which tools are appropriate:
  - E.g. modeling program behaviour vs. modeling the requirements
  - formal methods advocates get too attached to one tool!
- Formal methods require more effort
  - ...and the payoff is deferred

What are Formal Methods?

➜Broad View (Leveson)
- application of discrete mathematics to software engineering
- …involves modeling and analysis
- …with an underlying mathematically-precise notation

➜Narrow View (Wing)
- Use of a formal language
  - a set of strings over some well-defined alphabet, with rules for distinguishing which strings belong to the language
- Formal reasoning about formulae in the language
  - E.g. formal proofs: use axioms and proof rules to demonstrate that some formula is in the language

➜For requirements modeling…
- A notation is formal if:
  - …it comes with a formal set of rules which define its syntax and semantics.
  - …the rules can be used to analyse expressions to determine if they are syntactically well-formed or to prove properties about them.

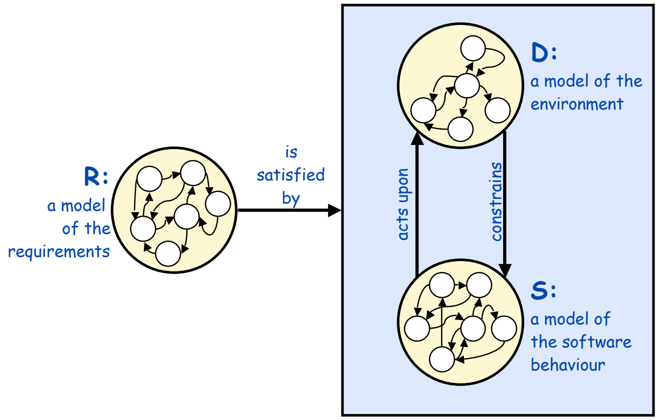
Varieties of formal analysis

➜ Consistency analysis and typechecking
- "Is the formal model well-formed?"
  - [assuming that we only use modeling languages where "well-formedness" is a useful thing to check]

➜ Validation:
- Animation of the model on small examples
- Formal challenges:
  - "if the model is correct then the following property should hold..."
- 'What if' questions:
  - reasoning about the consequences of particular requirements;
  - reasoning about the effect of possible changes
- State exploration
  - E.g. use a model checking to find traces that satisfy some property
- Checking application properties:
  - "will the system ever do the following..."

➜ Verifying design refinement
  - "does the design meet the requirements?"

Three different models??



University of Toronto Department of Computer Science

FM in practice

- From Shuttle Study [Crow & DiVito 1996]
  - More errors found in the process of formalizing the requirements than were found in the formal analysis
    - Formalization forces you to be precise and explicit, hence reveals problems
    - Formal analysis then finds fewer, but more subtle problems
  - Typical errors found include:
    - inconsistent interfaces
    - incorrect requirements (system does the wrong thing in response to an input)
    - clarity/maintainability problems

Issue Severity	*With FM*	*Existing*
High Major	2	0
Low Major	5	1
High Minor	17	3
Low Minor	6	0
Totals	30	4

© 2000-2003, Steve Easterbrook

University of Toronto Department of Computer Science

How do FMs differ?

→ Mathematical Foundation
- Logic
  - first order predicate logic - e.g. RML
  - temporal logic - e.g. Albert II, SCR, KAOS
  - multi-valued logic - e.g. Xchek
- Other
  - algebraic languages - e.g. Larch
  - set theory - e.g. Z

→ Ontology
- fixed
  - states, events, actions - e.g. SCR
  - entities, activities, assertions - e.g. RML
- extensible
  - meta language for defining new concepts - e.g. Telos

→ Treatment of Time
- State/event models
  - time as a discrete sequence of events - e.g. SCR
  - time as quantified intervals - e.g. KAOS
- Time as a first class object
  - meta-level class to represent time - e.g. Telos

© 2000-2003, Steve Easterbrook

University of Toronto Department of Computer Science

Three traditions ...

Formal Specification Languages
- Grew out of work on program verification
- Spawned many general purpose specification languages
  - Suitable for specifying the behaviour of program units
- Key technologies: Type checking, Theorem proving

Applicability to RE is poor
- No abstraction or structuring
- closely tied to program semantics

Examples: Larch, Z, VDM, …

Reactive System Modeling
- Grew out of a need to capture dynamic models of system behaviour
- Focus is on reactive systems (e.g. real-time, embedded control systems)
  - support reasoning about safety, liveness, performance(?)
  - provide a precise requirements specification language
- Key technologies: Consistency checking, Model checking

Applicability to RE is good
- modeling languages were developed specifically for RE

Examples: Statecharts, RSML, Parnas-tables, SCR, …

Formal Conceptual Modeling
- Grew out of a concern for capturing real-world knowledge in RE
- Focus is on modeling domain entities, activities, agents, assertions
  - provide a formal ontology for domain modeling
  - use first order predicate logic as the underlying formalism
- Key technologies: inference engines, default reasoning, KBS-shells

Applicability to RE is excellent
- modeling schemes capture key requirements concepts

Examples: Reqts Apprentice, RML, Telos, Albert II, …

© 2000-2003, Steve Easterbrook

(1) Formal *Specification* Languages

- Three basic flavours:
  - Operational - specification is executable abstraction of the implementation
    - good for rapid prototyping
    - e.g., Lisp, Prolog, Smalltalk
  - State-based - views a program as a (large) data structures whose state can be altered by procedure calls…
    - … using pre/post-conditions to specify the effect of procedures
    - e.g., VDM, Z
  - Algebraic - views a program as a set of abstract data structures with a set of operations…
    - … operations are defined declaratively by giving a set of axioms
    - e.g., Larch, CLEAR, OBJ
- Developed for specifying *programs*
  - Programs are formal, man-made objects
    - … and can be modeled precisely in terms of input-output behaviour
  - But in RE we're more concerned with:
    - real-world concepts, stakeholders, goals, loosely define problems, environments
  - So these languages are NOT appropriate for RE
    - but people fail to realise that requirements specification ≠ program specification

(2) Reactive System *Modeling*

- modeling how a system should behave
  - General approach:
    - Model the environment as a state machine
    - Model the system as a state machine
    - Model safety, liveness properties of the machine as temporal logic assertions
    - Check whether the properties hold of the system interacting with its environment
- Examples:
  - Statecharts
    - Harel's notation for modeling large systems
    - Adds parallelism, decomposition and conditional transitions to STDs
  - RSML
    - Heimdahl & Leveson's Requirements State Machine Language
    - Adds tabular specification of complex conditions to Statecharts
  - A7e approach
    - Major project led by Parnas to formalize A7e aircraft requirements spec
    - Uses tables to specify transition relations & outputs
  - SCR
    - Heitmeyer et. al. "Software Cost Reduction"
    - Extends the A7e approach to include dictionaries & support tables

(3) Formal Conceptual *Modeling*

- General approach
  - model the world beyond functional specifications
    - a specification is prescriptive, concentrating on desired properties of the machine
    - but we also need to capture an understanding of the application domain
    - hence build models of humans' knowledge/beliefs about the world
  - make use of abstraction & refinement as structuring primitives
- Examples:
  - RML - Requirements Modeling Language
    - Developed by Greenspan & Mylopoulos in mid-1980s
    - First major attempt to use knowledge representation techniques in RE
    - Essentially an object oriented language, with classes for activities, entities and assertions
    - Uses First Order Predicate Language as an underlying reasoning engine
  - Telos
    - Extends RML by creating a fully extensible ontology
    - meta-level classes define the ontology (the basic set is built in)
  - Albert II
    - developed by Dubois & du Bois in the mid-1990s
    - Models a set of interacting agents that perform actions that change their state
    - uses an object-oriented real-time temporal logic for reasoning

Using Formal Methods

- Selective use of Formal Methods
  - Amount of formality can vary
  - Need not build complete formal models
    - Apply to the most critical pieces
    - Apply where existing analysis techniques are weak
  - Need not formally analyze every system property
    - E.g. check safety properties only
  - Need not apply FM in every phase of development
    - E.g. use for modeling requirements, but don't formalize the system design
  - Can choose what level of abstraction (amount of detail) to model
- Lightweight Formal Methods
  - Have become popular as a means of getting the technology transferred
  - Two approaches
    - Lightweight *use of* FMs - selectively apply FMs for partial modeling
    - *Lightweight FMs* - new methods that allow unevaluated predicates