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 take more effort
  - They require 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 behavior 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 analyze expressions to determine if they are syntactically well-formed or to prove properties about them.

Varieties of formal analysis

→ Consistency analysis and type checking
  - “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??

- **R:** a model of the requirements
  - is satisfied by

- **S:** a model of the software behaviour

- **D:** a model of the environment

---

### 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 a first class object**
    - meta-level class to represent time - e.g. Telos

---

### 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

<table>
<thead>
<tr>
<th>Issue</th>
<th>Severity</th>
<th>With FM</th>
<th>Existing</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Major</td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Low Major</td>
<td></td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>High Minor</td>
<td></td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>Low Minor</td>
<td></td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>30</td>
<td>4</td>
</tr>
</tbody>
</table>

---

### 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

<table>
<thead>
<tr>
<th>Applicability to RE</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>Larch, Z, VDM</td>
</tr>
</tbody>
</table>

#### 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)
- Provide reasoning about safety, liveness, performance
- Key Technologies: Consistency checking, Model checking

<table>
<thead>
<tr>
<th>Applicability to RE</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>Statecharts, RSML, Parnas-tables, SCR</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Applicability to RE</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent</td>
<td>Reqs Apprentice, RML, Telos, Albert II</td>
</tr>
</tbody>
</table>
(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
% used to capture an understanding of the application domain
% hence build models of humans’ knowledge/beliefs about the world
% So these languages are NOT appropriate for RE

(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
% RSM:
  % Hoare & Leveson’s Requirements State Machine Language
  % Adds table specification of complex conditions to Statecharts
% ATe approach:
  % Major project led by Parnas to formalize ATe aircraft requirements spec
  % Uses tables to specify transition relations & outputs
% SCR
  % Heitmeyer et. al. “Software Cost Reduction”
  % Extends the ATe 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 unvaluated predicates