Process Models: Specialized Process Models (Continue)

LECTURE # 9

Lecture by Engr. Sidra

Advantages of CBD

- Faster development
- Easier to maintain
- Increase Quality
- Easier to create applications variants and upgrades
- Lower overall development cost

Disadvantages of CBD

- Component trustworthiness - how can a component with no available source code be trusted?
- Component certification - who will certify the quality of components?
- Requirements trade-offs - how do we do trade-off analysis between the features of one component and another?
- Choosing Middleware - Incompatible technologies
- Testing is Harder - Unknown uses of components
- High initial cost - Training of developers

Specialized Process model

- Special process models take many features from one or more conventional models.
- However these special models tend to be applied when a narrowly defined software engineering approach is chosen.
- Types in Specialized process models:
  - 1. Component based development (Promotes reusable components)
  - **2. The formal methods model (Mathematical formal methods are backbone here)**
  - 3. Aspect oriented software development (Uses crosscutting technology)
  - Unified Process (use-case driven, architecture centric)

What are Formal Methods?

- Broad View
  - application of discrete mathematics to software engineering
  - involves modeling and analysis
  - with an underlying mathematically-precise notation
- Narrow View
  - 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

Formal Methods in Software Engineering

What to formalize?

- models of requirements knowledge (so we can reason about them)
- specifications of requirements (so we can document them precisely)
- Specifications of program design (so we can verify correctness)

- **Why formalize?**
- Removes ambiguity and improves precision
- To verify that the requirements have been met
- To reason about the requirements/designs
  - Properties can be checked automatically
  - Test for consistency, explore consequences, etc.
- To animate/execute specifications
  - Helps with visualization and validation
- …because we have to formalize eventually anyway
  - Need to bridge from the informal world to a formal machine domain

Formal Methods Model

- The **formal methods model** is concerned with the application of a mathematical technique to design and implement the software.
- Formal methods are techniques used to model complex systems as mathematical entities. By building a mathematically rigorous model of a complex system, it is possible to verify the system's properties in a more thorough fashion than empirical testing.
- The formal methods used during the development process provide a mechanism for eliminating problems, which are difficult to overcome using other software process models.
- The software engineer creates formal specifications for this model. These methods minimize specification errors and this result in fewer errors when the user begins using the system.

Formal Methods Model

- For requirements modeling…
  - A notation is formal if:
    - it comes with a formal set of rules which define its syntax and semantics.
- Formal specification is expressed in a language whose syntax and semantics are formally defined.
- This language comprises a syntax that defines specific notation used for specification representation; semantic, which uses objects to describe the system; and a set of relations, which uses rules to indicate the objects for satisfying the specification.

Examples

1. **A>B and C>D**
2. *exists* ***i, j, k*** *in* **M...N:** $i^2 = j^2 + k^2$
3. *for-all i* in **1...10**, exists *j* in **1...10**: squares $(i)=j^2$

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Two Approaches: Property-Based Specification

- It describes the operations performed on the system.
- In addition, it describes the relationship that exists among these operations.
- A property-based specification consists of two parts:
  - signatures, which determine the syntax of operations and
  - an equation, which defines the semantics of the operations through a set of equations known as **axioms.**

```
AIRCRAFT_TABLE(AIRCRAFT_DETAILS)
sort aircraft_table
imports integer, aircraft_track, aircraft_details
```

Two Approaches: Property-Based Specification

- For example, consider a simple Instant Messaging application for your cell phone. Then some operations might be:
  - Start up, Send message, Receive message, Display message, Shut down
- The relationships between these operations might include:
  - Startup must come before any other operation
  - Shut down must be the last operation performed
  - Display message comes during each send message and after each receive message

Example

SIGNATURE

create(integer) → aircraft_table

insert(aircraft_table, aircraft_track, aircraft_details) → aircraft_table

remove(aircraft_table, aircraft_track) → aircraft_table

size(aircraft_table) → integer

eval(aircraft_table, aircraft_track) → aircraft_details

AXIOM

```
size( create(i) ) = i
size( insert(x, n, y) ) = size( x ) + 1
size( remove(x, m) ) = size( x ) - 1
eval( create(i), n) = Undefined
eval( insert(x, n, y), m) =
                    if m == n
                      then y
                    else
                      then eval( x, m )
eval( remove(x,n), m) =
                    if m == n
                      then Undefined
                    else
                      then eval( x, m )
```

where

```
x   is an   aircraft_table
n   is an   aircraft_track
m   is an   aircraft_track
y    are    aircraft_details
```

Two Approaches: Model-Based Specification

- It utilizes the tools of set theory, function theory, and logic to develop an abstract model of the system.
- In addition, it specifies the operations performed on the abstract model.
- A model-based specification comprises
  - a definition of the set of states of the system and
  - definitions of the legal operations performed on the system to indicate how these legal operations change the current state.

Two Approaches: Model-Based Specification

- Consider the Instant Messaging application example mentioned above. States the system may be in might include:
  - Starting up, Sending message, Receiving message, Displaying message, Shutting down
- As for transitions, they might include:
  - Clicking the application icon to enter starting up
  - Or pressing the send button to leave the sending message state

Message_connection:M system _ Connections
Assigned to: instant_connection: system_connection

Example

INIT( )
ext **wr** inflight : *Aircraft*-set
wr onground : *Aircraft*-set
post inflight = { }
onground = { }

ENTERSPACE(a : *Aircraft*)
ext **wr** inflight : *Aircraft*-set
rd onground : *Aircraft*-set
pre $a \notin$ inflight $\wedge$ a $\notin$ onground
post inflight = inflight' $\cup$ {a}