Lecture 10, Part 1: Verification and Validation

Jennifer Campbell
CSC340 - Winter 2007

UML Models

• Activity diagrams
  – capture business processes involving concurrency and synchronization
  – good for analyzing dependencies between tasks
• Class Diagrams
  – capture the structure of the information used by the system
  – good for analyzing the relationships between data items used by the system
  – good for helping you identify a modular structure for the system
• Statecharts
  – capture all possible responses of an object to all use cases in which it is involved
  – good for modeling the dynamic behavior of a class of objects
  – good for analyzing event ordering, reachability, deadlock, etc.

Use Cases
• capture the view of the system from the view of its users
• good starting point for specification of functionality
• good visual overview of the main functional requirements

Sequence Diagrams
• capture an individual scenario (one path through a use case)
• good for modeling dialog structure for a user interface or a business process
• good for identifying which objects (classes) participate in each use case
• helps confirm that all the necessary classes and operations have been identified

Objectives of V&V
“The overall objective of V&V approaches is to insure that the project is free from failures and meets its user’s expectations.”

Correctness
– The product is free of errors.

Consistency
– The product is consistent (within itself and with other related products).

Necessity
– Everything in the product is necessary.

Sufficiency
– The product is complete.

Quality
– The product satisfies its quality requirements.

[Col88]
Inquiry Cycle

Prior Knowledge (e.g. customer feedback)

Observe (what is wrong with the current system?)

Model (describe/explain the observed problems)

Design (invent a better system)

Intervene (replace the old system)

Note similarity with process of scientific investigation: Requirements models are theories about the world; Designs are tests of those theories

Shortcuts in the inquiry cycle

Prior Knowledge (e.g. customer feedback)

Observe (what is wrong with the current system?)

Model (describe/explain the observed problems)

Design (invent a better system)

Intervene (replace the old system)

Build a Prototype

Get users to try it

Analyze the model

Check properties of the model

Create/refine a better theory

Initial hypotheses

Look for anomalies - what can't the current theory explain?

Create/refine a better theory

Design experiments to test the new theory

Design

Invent a better system

Interpret replace the old system

Requirements models are theories about the world; Designs are tests of those theories

Refresher: V&V Criteria

- **Domain Properties**: things in the application domain that are true anyway
- **Requirements**: things in the application domain that we wish to be made true
- **Specification**: a description of the behaviours the program must have in order to meet the requirements

- **Two verification criteria**:
  - The Program running on a particular Computer satisfies the Specification
  - The Specification, given the Domain properties, satisfies the Requirements

- **Two validation criteria**:
  - Did we discover (and understand) all the important Requirements?
  - Did we discover (and understand) all the relevant Domain properties?

V&V Example

- **Requirement R**: “Reverse thrust shall only be enabled when the aircraft is moving on the runway”
- **Domain Properties D**:
  - Wheel pulses on if and only if wheels turning
  - Wheels turning if and only if moving on runway
- **Specification S**:
  - Reverse thrust enabled if and only if wheel pulses on
- **Validation**
  - Are our assumptions, D, about the domain correct? Did we miss any?
  - Are the requirements, R, what is really needed? Did we miss any?
- **Verification**
  - Does the flight software, P, running on the aircraft flight computer, C, correctly implement S?
  - Does S, in the context of assumptions D, satisfy R?

V&V Activities

- **Reviews**
  - Walkthroughs, inspections, etc.
- **Software Testing**
  - Not applicable to RE.
- **Formal Methods**
  - Use mathematics to prove that the requirements are consistent.
- **Consistency checking** (this can also be done formally)
  - Verifying consistency between models
- **Prototyping**
  - Present a prototype to the stakeholder to confirm that it has the expected behaviour.
- **Requirements Tracing**
  - Trace each requirement back to its source.

V&V Activities: Reviews

(Fagan) Inspections
- a process management tool
- always formal
- used to improve quality of the development process
- collect defect data to analyze the quality of the process
- written output is important
- major role in training junior staff and transferring expertise

Management reviews
- E.g. preliminary design review (PDR), critical design review (CDR), ...
- Used to provide confidence that the requirements are sound
- Attended by management and sponsors (customers)
- Often just a “dog-and-pony show”

Walkthroughs
- developer technique (usually informal)
- used by development teams to improve quality of product
- focus is on finding defects

Review the SRS with stakeholders to validate.
**V&V Activities: Formal Methods**

**Model Analysis**
- 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
    - "will the system ever do the following..."
  - State exploration
    - E.g. use a model checking to find traces that satisfy some property
    - "Is the model well-formed?"
    - Are the parts of the model consistent with one another?

**Verification**

**V&V Activities: Consistency Checking [2]**

**Use Case Diagrams**
- Does each use case have a user?
- Does each use case have at least one use case?

**Sequence Diagrams**
- Is each use case documented?
- Using sequence diagrams or equivalent

**V&V Activities: Consistency Checking [3]**

**Sequence Diagrams**
- Is each class in the class diagram?
- Can each message be sent?
  - Is there an association connecting sender and receiver classes on the class diagram?
  - Is there a method call in the sending class for each sent message?
  - Is there a method call in the receiving class for each received message?

**Class Diagrams**
- Does each statechart diagram capture the states of a single class?
- Does each transition have a trigger event?
  - Is it clear which object initiates each event?
  - Is each event listed as an operation for that object’s class in the class diagram?
- Does each state represent a distinct combination of attribute values?
  - Is it clear which combination of attribute values?
  - Are all those attributes shown on the class diagram?
- Are there method calls in the class diagram for each transition?
  - ...a method call that will update attribute values for the new state?
  - ...method calls that will test any conditions on the transition?
  - ...method calls that will carry out any actions on the transition?

**V&V Activities: Consistency Checking [4]**

**Class Diagrams**
- Does each statechart diagram capture the states of a single class?

**Statechart Diagrams**
- Does each statechart diagram capture the states of a single class?
- Does each transition have a trigger event?
  - Is it clear which object initiates each event?
  - Is each event listed as an operation for that object’s class in the class diagram?
- Does each state represent a distinct combination of attribute values?
  - Is it clear which combination of attribute values?
  - Are all those attributes shown on the class diagram?
- Are there method calls in the class diagram for each transition?
  - ...a method call that will update attribute values for the new state?
  - ...method calls that will test any conditions on the transition?
  - ...method calls that will carry out any actions on the transition?

**V&V Activities: Prototyping**

"A software prototype is a partial implementation constructed primarily to enable customers, users, or developers to learn more about a problem or its solution."

[Davis 1990]

"Prototyping is the process of building a working model of the system"

[Agresti 1986]
V&V Activities: Prototyping [2]

Approaches to prototyping

- **Presentation Prototypes**
  - explain, demonstrate and inform – then throw away
  - e.g. used for proof of concept; explaining design features; etc.

- **Exploratory Prototypes**
  - used to determine problems, elicit needs, clarify goals, compare design options
  - informal, unstructured and thrown away.

- **Breadboards or Experimental Prototypes**
  - explore technical feasibility; test suitability of a technology
  - Typically no user/customer involvement

- **Evolutionary (e.g. “operational prototypes”, “pilot systems”)**
  - development seen as continuous process of adapting the system
  - “prototype” is an early deliverable, to be continually improved.

---

V&V Activities: Prototyping [3]

**Throwaway Prototyping**
- **Purpose:**
  - to learn more about the problem or its solution...
  - discard after desired knowledge is gained.

- **Use:**
  - early or late

- **Approach:**
  - horizontal - build only one layer (e.g. UI)
  - “quick and dirty”

**Evolutionary Prototyping**
- **Purpose**
  - to learn more about the problem or its solution...
  - …and reduce risk by building parts early

- **Use:**
  - incremental; evolutionary

- **Approach:**
  - vertical - partial impl. of all layers;
  - designed to be extended/adapted

---

V&V Activities: Tracing [2]

**Forward traceability:** trace requirements from stakeholders to requirements specification.

**Traceability matrix:**

<table>
<thead>
<tr>
<th>ID</th>
<th>User Requirements</th>
<th>Forward Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>Users shall process retirement claims.</td>
<td>R10, R11, R12</td>
</tr>
<tr>
<td>S3</td>
<td>Users shall process survivor claims</td>
<td>R13</td>
</tr>
</tbody>
</table>

---

V&V Activities: Tracing [4]

**Backward traceability:** trace requirements from req spec to stakeholder. Traceability matrix:

<table>
<thead>
<tr>
<th>ID</th>
<th>User Requirements</th>
<th>Backward Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>R10</td>
<td>The system shall accept requirement data.</td>
<td>S2</td>
</tr>
<tr>
<td>R11</td>
<td>The system shall calculate the amount of retirement.</td>
<td>S2</td>
</tr>
<tr>
<td>R12</td>
<td>The system shall calculate point-to-point travel time.</td>
<td>S2</td>
</tr>
<tr>
<td>R13</td>
<td>The system shall calculate the amount of survivor annuity.</td>
<td>S3</td>
</tr>
</tbody>
</table>

---

Independent V&V

- V&V performed by a separate contractor

- Three types of independence:
  - **Managerial Independence:**
    - separate responsibility from that of developing the software
    - can decide when and where to focus the V&V effort
  - **Financial Independence:**
    - Costed and funded separately
    - No risk of diverting resources when the going gets tough
  - **Technical Independence:**
    - Different personnel, to avoid analyst bias
    - Use of different tools and techniques
References


