Validation Techniques

• What are goals of verification and validation?
• Checking quality
• Model analysis
• Prototyping

Refresher: V&V Criteria

• Some distinctions:
  ➢ 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?
Validation Techniques

- What are goals of verification and validation?
- **Checking quality**
- Model analysis
- Prototyping
Quality of Requirements Specification

J. Krogstie, Semiotic Quality Framework

SEQUAL

- **L** - language extension
- **D** - the domain
- **M** - externalised model
- **K_s** - relevant explicit knowledge of the stakeholders
- **K_m** - relevant explicit knowledge of modellers
- **I** - social actor interpretation
- **T** - technical actor interpretation
- **G** - organisationally-motivated goals of the modelling task
Physical quality

- Although information systems specifications and models are not of the physical kind, any model can be represented physically - e.g. on disk or paper

- Specification should be
  - Electronically stored
Empirical quality

→ **Empirical** quality deals with the variety of elements distinguished, error frequencies when being written or read, coding and ergonomics for computer-human interaction, for documentation and tools.

- Specification should be:
  - **Understandable**
    - all classes readers can easily comprehend the meaning of all requirements with a minimum of explanation.
  - **Concise**
    - it is as short as possible without affecting any other quality of the requirements specification.

---

Pragmatic quality

- **Pragmatic** quality is the correspondence between the model and the audience’s interpretation of it.

- Specification should be:
  - **Executable/Interpretatable/Prototypable**
    - there exists a software tool capable of inputting the requirements specification and providing a dynamic behavioural model.
  - **Organised**
    - its contents are arranged so that readers can easily locate information and logical relationships among adjacent sections are apparent.
  - **Cross-referenced**
    - Cross-references are used to relate sections containing requirements with other relative requirements.
Social quality

• The goal for social quality is agreement.

• Tool support
  • models created based on the different internal reality of the participants that are to agree

Syntactic quality

→ Syntactic quality includes correctness of lexicon, syntax and structural quality.

• Syntactic errors:
  • Syntactic invalidity
  • Syntactic incompleteness

• Automated tool support:
  • Error prevention
  • Error detection
  • Error correction
Semantic quality

→ **Semantic** quality is the correspondence between
the model and the modelling domain

→ **Feasibility:**

→ Attempts at reaching a state of total validity and
completeness will lead to unlimited spending of time and
money on the modelling activity.

→ The time to terminate a modelling activity is thus not when
the model is “**perfect**” (**which will never happen**), but
when it has reached a state where further modelling is
regarded to be less beneficial than applying the model in
its current state.

---

Semantic quality

- **Complete**
  - everything that the software is supposed to do is included;
  - responses of the software to all realisable classes of input data in all
    recognisable classes of situations are included;
  - all pages are numbered, all figures and tables are numbered,
    named, and referenced; all terms are defined; all units of measure
    are provided; and all referenced material are present;
  - no sections are marked "To be determined".

- **Correct**
  - every requirement represent something required of the system to be
    built.

  **User needs** that the system **must respond** button press within 5
  seconds.

  **Requirement:** the system **must respond** button press within 10 seconds.
Semantic quality

• Internally consistent
  • no subset of individual requirements stated therein conflict
    
    a) The light shall be lit when the button is pressed.
    b) When the button is released, the light shall become lit.

    a) The system should prompt the message
    b) The system should cue the message

• Externally consistent
  • no requirement stated therein conflict with any already base-lined project documentation

Semantic quality

• Annotated by
  • relative importance, relevant stability, version

• Precise
  • (1) numeric quantities are used whenever possible;
  • (2) the appropriate levels of precision are used for all numeric quantities.

• Traced
  • the origin of each of its requirements is clear

  The system shall respond to any occurrence of request X within 20 seconds.

• Traceable
  • it is written in a manner that facilitates the referencing of each individual statement

• Verifiable
  • there exist finite, cost effective techniques that can be used to verify that every requirement stated therein is satisfied by the system to be built.
Semantic quality

• Achievable
  • there could exist at least one system design and implementation that correctly implements all the requirements stated in the requirements specification

• Design-independent
  • there exist more than one system design and implementation that correctly implements all requirements stated in the requirements specification

• At the right level of detail
  • specific enough so that any system built that satisfies the requirements in the specification satisfies all user needs
  • abstract enough so that all systems that satisfy all user needs also satisfy all requirements

Semantic quality

• Modifiable
  • structure and style are such that any changes can be made easily, completely and consistently

• Unambiguous
  • every requirement stated therein has only one possible interpretation

Up to 12 aircraft, the small display format shall be used. Otherwise the large display format shall be used.

Aircraft that are non-friendly and have an unknown mission or the potential to enter restricted airspace within 5 minutes shall rise an alert.

Aircraft that are non-friendly and have an unknown mission or the potential to enter restricted airspace within 5 minutes shall rise an alert.

Aircraft that are either non-friendly and have an unknown mission or have the potential to enter restricted airspace within 5 minutes shall rise an alert.

Aircraft that are non-friendly and either have an unknown mission or the potential to enter restricted airspace within 5 minutes shall rise an alert.
Orthogonal aspects

- **Reusable**
  - its sentences, paragraphs, and sections can be easily adopted and adapted for use in subsequent requirements specification.

- **Physical quality**
  - a persistent form that is available to those who potentially will want to reuse it

- **Syntactic quality**
  - syntactically correct

- **Semantic quality**
  - Similar domains
  - white-box reuse - modifiable, comprehensible and comprehended (techniques of pragmatic quality), annotated, and other

- **Social quality**
  - model integration and conflict resolution can be useful to investigate to what extent the solutions based on the model to be reused, should be reused.

---

**THERE IS NO SUCH THING AS A PERFECT REQUIREMENTS SPECIFICATION!**

- ambiguous
  - expand
  - condense
- redundant
  - reduce
  - add explanations
- not understandable
  - formalise
- inconsistent
  - resolve
- incomplete
  - expand
  - formalise
Validation Techniques

- What are goals of verification and validation?
- Checking quality
- Model analysis
- Prototyping

Model Checking

- Has revolutionized formal verification:
  - emphasis on partial verification of partial models
    - E.g. as a debugging tool for state machine models
  - fully automated
- What it does:
  - Mathematically – computes the “satisfies” relation:
    - Given a temporal logic theory, checks whether a given finite state machine is a model for that theory.
  - Engineering view – checks whether properties hold:
    - Given a model (e.g. a FSM), checks whether it obeys various safety and liveness properties
- How to apply it in RE:
  - The model is an (operational) Specification
    - Check whether particular requirements hold of the spec
  - The model is (an abstracted portion of) the Requirements
    - Carry out basic validity tests as the model is developed
  - The model is a conjunction of the Requirements and the Domain
    - Formalise assumptions and test whether the model respects them
Model Analysis

• **Verification**
  - “Is the model well-formed?”
  - Are the parts of the model consistent with one another?

• **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
      - “will the system ever do the following…”
  - State exploration
    - E.g. use a model checking to find traces that satisfy some property

Requirements Specification

1 Introduction
   - Purpose
   - Scope
   - Definitions, acronyms, abbreviations
   - Reference documents
   - Overview

2 Overall Description
   - Product perspective
   - Product functions
   - User characteristics
   - Constraints
   - Assumptions and Dependencies

3 Specific Requirements
Appendices
Index
Goal-Scenario coupling

... initiate and influence the definition of ...
... classify ...

Goals

Scenarios

... illustrate satisfaction...
... lead to the identification of new ...
... lead to revision of ...

Key Relationships

Elicitation, refinement and validation of solution-oriented requirements

Solution oriented requirements

Refinement of existing and elicitation of new goals and scenarios
We’ve looked at the following non-UML diagrams

➤ **Goal Models**
- Capture strategic goals of stakeholders
- Good for exploring 'how' and 'why' questions with stakeholders
- Good for analysing trade-offs, especially over design choices

➤ **Strategic Dependency Models (i†)**
- Capture relationships between actors in an organisational setting
- Helps to relate goal models to organisational setting
- Good for understanding how the organisation will be changed
### Use cases

**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

**Cross-checks:**
- Does each use case have a user?
  - Does each user have at least one use case?
- Is each use case documented?
  - Using sequence diagrams or use case template

### Class diagrams

**Class Diagrams**
- capture the structure of the information used by the system
- good for analysing the relationships between data items used by the system
- good for helping you identify a modular structure for the system

**Cross checks**
- Does the class diagram capture all the classes mentioned in
  - other diagrams?
  - specification glossary?
- Does every class have methods to get/set its attributes?
**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.

**Cross-checks:**
- Does each statechart diagram capture (the states of) a single class?
  - Is that class in the class diagram?
- 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?

**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 you check that you identified all the necessary classes and operations

**Cross-checks:**
- 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?
Validation Techniques

- What are goals of verification and validation?
- Checking quality
- Model analysis
- **Prototyping**

Prototyping lifecycle

- **Prototyping is used for:**
  - understanding the requirements for the user interface
  - examining feasibility of a proposed design approach
  - exploring system performance issues

- **Problems:**
  - users treat the prototype as the solution
  - a prototype is only a partial specification
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]

• 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.

• 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”
  ➢ Advantages:
    • Learning medium for better convergence
    • Early delivery → early testing → less cost
    • Successful even if it fails!
  ➢ Disadvantages:
    • Wasted effort if reqts change rapidly
    • Often replaces proper documentation of the requirements
    • May set customers’ expectations too high
    • Can get developed into final product

• 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
  ➢ Advantages:
    • Requirements not frozen
    • Return to last increment if error is found
    • Flexible(?)
  ➢ Disadvantages:
    • Can end up with complex, unstructured system which is hard to maintain
    • early architectural choice may be poor
    • Optimal solutions not guaranteed
    • Lacks control and direction
Validation Techniques

- What are goals of verification and validation?
- Checking quality
- Model analysis
- Prototyping