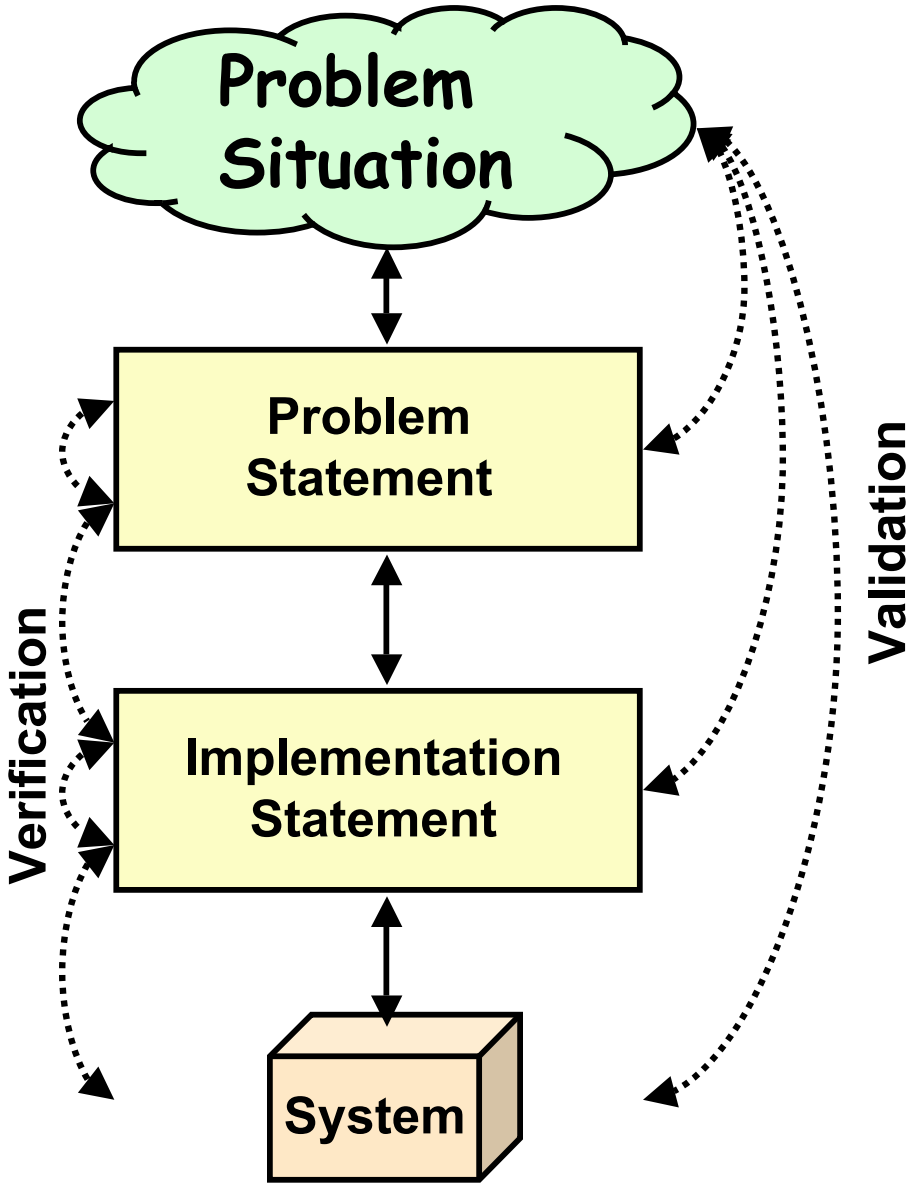
Lecture 18: Verification and Validation

- Refresher:
  - definitions for V&V
- Validation Techniques
  - Prototyping
  - Model Analysis (e.g. Model Checking)
  - Inspection
- Verification Techniques
  - Making Specifications Traceable (see lecture 20)
  - Testing (not covered in this course)
  - Code Inspection (not covered in this course)
  - Code analysis (not covered in this course)
- Independent V&V

Verification and Validation

- **Validation:**
  - "Are we building the right system?"
  - Does our problem statement accurately capture the real problem?
  - Did we account for the needs of all the stakeholders?
- **Verification:**
  - "Are we building the system right?"
  - Does our design meet the spec?
  - Does our implementation meet the spec?
  - Does the delivered system do what we said it would do?
  - Are our requirements models consistent with one another?



Refresher: V&V Criteria

Source: Adapted from Jackson, 1995, p170-171

→ 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

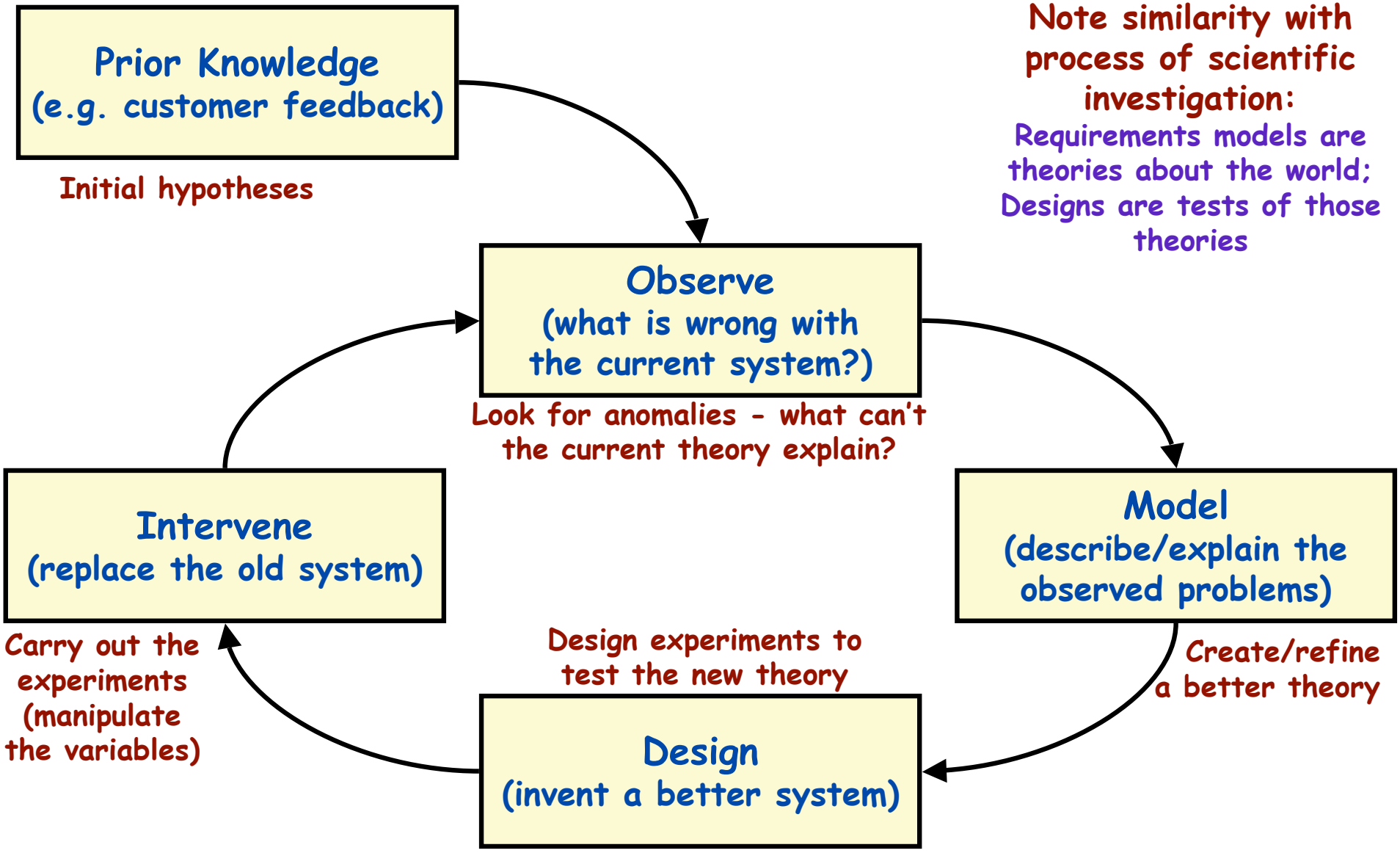
→ Two validation criteria:

- Did we discover (and understand) all the important Requirements?
- Did we discover (and understand) all the relevant Domain properties?

V&V Example

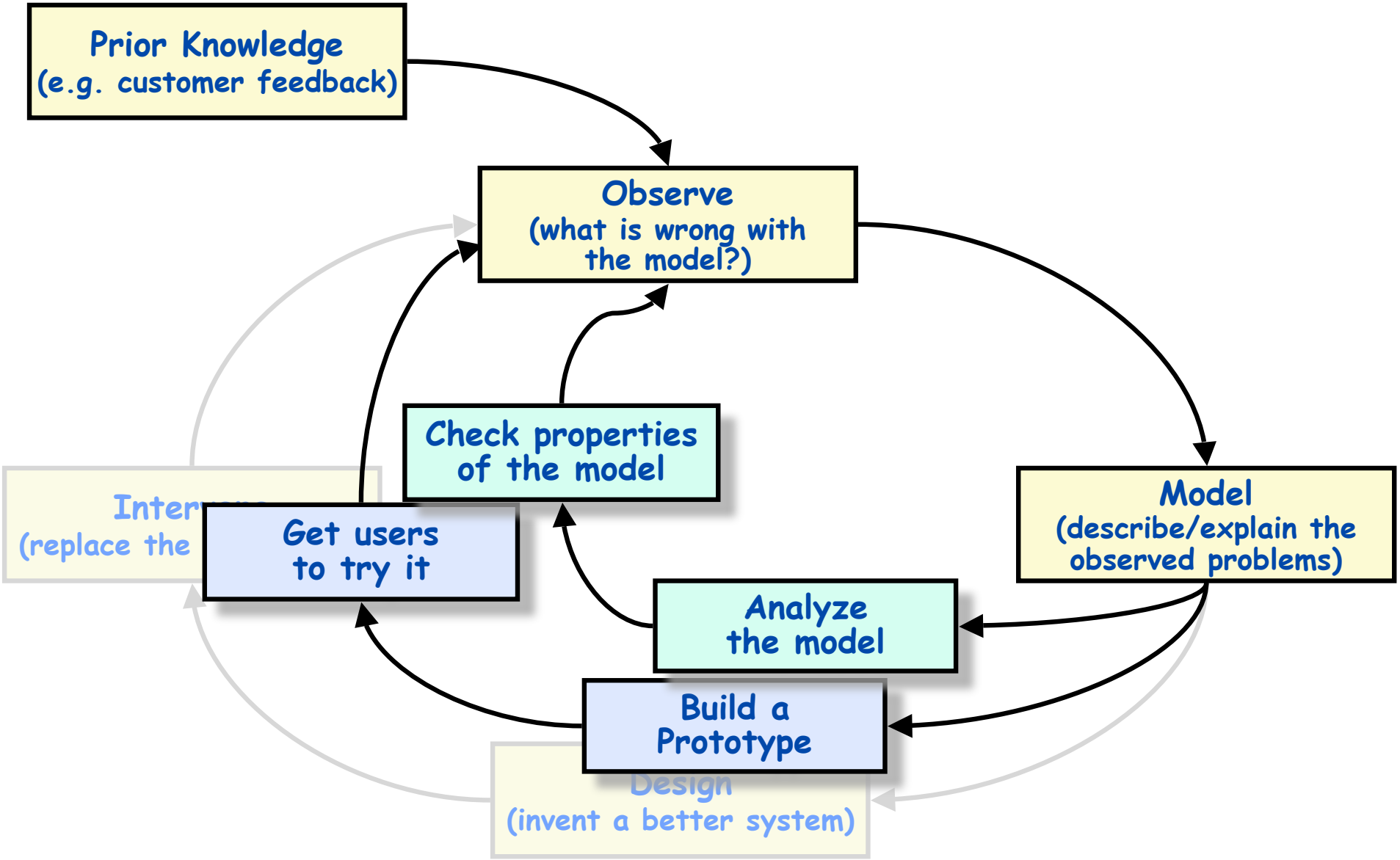
- Example:
  - Requirement R:
    - "Reverse thrust shall only be enabled when the aircraft is moving on the runway"
  - Domain Properties D:
    - Wheel pulses on if and only if wheels turning
    - Wheels turning if and only if moving on runway
  - Specification S:
    - Reverse thrust enabled if and only if wheel pulses on
- 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?
- 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?

Inquiry Cycle



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



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
  - used for proof of concept; explaining design features; etc.
  - explain, demonstrate and inform – then throw away
- 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 or Evolve?

→ 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

Brooks: "Plan to throw one away - you will anyway!"

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 model checking to find traces that satisfy some property

Basic Cross-Checks for UML

Use Case Diagrams

- 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 equivalent

Class Diagrams

- Does the class diagram capture all the classes mentioned in other diagrams?
- Does every class have methods to get/set its attributes?

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?

StateChart Diagrams

- 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?

Reviews, Walkthroughs, Inspections…

→ "Management reviews"
- E.g. preliminary design review (PDR), critical design review (CDR), …
- Used to provide confidence that the design is 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

→ "(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

→ These definitions are not widely agreed!
- Other terms used:
  - Formal Technical Reviews (FTRs)
  - Formal Inspections

→ "Formality" can vary:
- informal:
  - meetings over coffee,
  - regular team meetings,
  - etc.
- formal:
  - scheduled meetings,
  - prepared participants,
  - defined agenda,
  - specific format,
  - documented output

Benefits of formal inspection

Source: *Adapted from Blum, 1992, Freedman and Weinberg, 1990, & notes from Philip Johnson.*

- Formal inspection works well for programming:
  - For applications programming:
    - more effective than testing
    - most reviewed programs run correctly first time
    - compare: 10-50 attempts for test/debug approach
  - Data from large projects
    - error reduction by a factor of 5; (10 in some reported cases)
    - improvement in productivity: 14% to 25%
    - percentage of errors found by inspection: 58% to 82%
    - cost reduction of 50%-80% for V&V (even including cost of inspection)
  - Effects on staff competence:
    - increased morale, reduced turnover
    - better estimation and scheduling (more knowledge about defect profiles)
    - better management recognition of staff ability
- These benefits also apply to requirements inspections
  - Many empirical studies investigated variant inspection processes
  - Mixed results on the relative benefits of different processes

Roles

Source: *Adapted from Blum, 1992, pp369-373*

Formal Walkthrough

- **Review Leader**
  - chairs the meeting
  - ensures preparation is done
  - keeps review focussed
  - reports the results
- **Recorder**
  - keeps track of issues raised
- **Reader**
  - summarizes the product piece by piece during the review
- **Author**
  - should actively participate (e.g. as reader)
- **Other Reviewers**
  - task is to find and report issues

Fagan Inspection

- **Moderator**
  - must be a competent programmer
  - should be specially trained
  - could be from another project
- **Designer**
  - programmer who produced the design being inspected
- **Coder/Implementor**
  - programmer responsible for translating the design to code
- **Tester**
  - person responsible for writing/executing test cases

Structuring the inspection

- Checklist
  - uses a checklist of questions/issues
  - review structured by issue on the list
- Walkthough
  - one person presents the product step-by-step
  - review is structured by the product
- Round Robin
  - each reviewer in turn gets to raise an issue
  - review is structured by the review team
- Speed Review
  - each reviewer gets 3 minutes to review a chunk, then passes to the next person
  - good for assessing comprehensibility!

Why use inspection?

- Inspections are very effective
  - *Code* inspections are better than testing for finding defects
  - For *Specifications*, inspection is all we have (you can't "test" a spec!)
- Key ideas:
  - Preparation: reviewers inspect individually first
  - Collection meeting: reviewers meet to merge their defect lists
  - Log each defect, but don't spend time trying to fix it
  - The meeting plays an important role:
    - Reviewers learn from one another when they compare their lists
    - Additional defects are uncovered
  - Defect profiles from inspection are important for process improvement
- Wide choice of inspection techniques:
  - What roles to use in the meeting?
  - How to structure the meeting?
  - What kind of checklist to use?

Independent V&V

- V&V performed by a separate contractor
  - Independent V&V fulfills the need for an independent technical opinion.
  - Cost between 5% and 15% of development costs
  - Studies show up to fivefold return on investment:
    - Errors found earlier, cheaper to fix, cheaper to re-test
    - Clearer specifications
    - Developer more likely to use best practices
- 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

Summary

- **Validation** checks you are solving the right problem
  - Prototyping - gets customer feedback early
  - Inspection - domain experts read the spec carefully
  - Formal Analysis - mathematical analysis of your models
  - …plus meetings & regular communication with stakeholders
- **Verification** checks your engineering steps are sound
  - Consistency checking - do the models agree with one another?
  - Traceability - do the design/code/test cases reflect the requirements?
- Use appropriate V&V:
  - Early customer feedback if your models are just sketches
  - Analysis and consistency checking if your models are specifications
  - Independence important if your system is safety-critical