A. What is Verification & Validation

- Validation = are we building the right *product*?
  - Customer focus
  - Treated at either end of lifecycle
- Verification = are we building the product *right*?
  - Process view
  - Transformational view of software development
  - Product = $T_n(...T_3(T_2(T_1, R_0) ...)$
    - Verification: is $R_k$ equivalent to $R_{k-1}$
    - Verification is incremental: Is each transformation $T_i$ correct?
- V&V recognize that software development is an error prone process
  - Catch errors as soon as possible
  - Self-correcting process: $V_k(T_k(R_{k-1}))$ is equivalent to $R_{k-1}$
  - Verification itself is error prone
  - Redundancy Principle: multiple viewpoints decrease chance of missing errors
    - Cost-benefit trade-off
- Representation is the central issue in V&V
  - Requirements gathering is a fundamental problem
  - Specification is a fundamental problem in software engineering
  - Validation requires both the capture of requirements and their correct specification to software developers
  - Verification compares a representation at specificity level L with representations at specification levels less than L
  - The language used in a representation must be well defined
    - Lexical, syntactic and semantic rules
    - Verification rules that relate to representations of lower (or higher) specification levels
    - Well defined rules facilitate automation of transformation and verification
  - Formal Methods provide representation languages with these properties
  - Software engineering can be viewed as a database problem
    - All representations contain/communicate relationships between entities in the representation
    - Graphical representations have an underlying database representation
    - Automation applies to the underlying representations
B. Validation Issues

- Validation is difficult because requirements gathering and requirements specification are both hard software engineering problems
- Ultimately, validation is a contract between producer and consumer
  - Initially, the specification is agreed to be equivalent to the consumer's requirements (desires)
    - Human reasoning used to establish the equivalence
    - Documentation is used to make the equivalence explicit – traceability
    - Natural language augmented by formal (mathematical, well-defined) models
    - NOTE: possibility of knowledge gaps between the representations
  - Finally, the product is shown to satisfy all the requirements – acceptance testing
    - Test cases linked to the specification: indirect testing
    - Test cases linked to the requirements (higher level of abstraction)
- Knowledge shift in the time between requirements gathering/specification and software development
  - Requirements changes
    - Knowledge gaps widen (small unknowns may become major surprises)
  - The contract evolves – changes must be tracked
  - Technology barriers discovered

C. Verification Issues

- Team verification process
  - Inspections / walkthroughs
  - Applicable across the lifecycle
- Effectiveness
  - Human verification very effective
  - Ability to read between the lines an important aspect to human verification
    - Testing rarely identifies missing pieces, i.e., sins of omission
    - Expertise and experience pay dividends – involve people with diverse key skills
- Efficiency/Economy
  - Duplication is not redundancy – duplication wastes effort, redundancy provides multiple insights
  - Stratified inspection (author's term)
    - Each reviewer reviews a specified layer of meaning, e.g., lexical, syntactic, semantic
    - Minimal duplication
    - Views integrated during the inspection process
- Team composition
  - 1-person reviews => lowest cost, low (often insufficient) redundancy
  - n-person reviews => what value of n?
  - n-person, mixed roles
D. Software Testing – Dynamic Verification

- There is a test process that mirrors the software development process
  - Analysis – understand the software being tested, set strategy
  - Design – apply some technique to create test cases based on strategy
  - Implementation – develop the machinery to execute the tests
    - Manual test scripts *(human in the loop)*
    - Automated test programs (e.g., drivers) to execute tests using stored data and to save results
    - Automated tools to generate test scripts and drivers
  - Execution – perform the defined tests
  - Evaluation – identify and handle defects
    - Mismatch between expected and actual result
      - Defect in the test case, test script, tester, test program, or program under test??
      - Test scripts/programs must be *debugged*
    - Defect classification
      - Type of defect
      - When introduced
      - Cost to fix => severity
      - Root cause
    - Problem Tracking
    - Defect removal
      - Change control
    - Re-Test (regression test)
  - Process improvement
    - Revise transformation to prevent the defect
    - Revise the verification process to detect the defect
    - Cost-driven process improvement (biggest bang for buck)
- Verification of the test process??
  - Yes! *Are we testing the product?*
  - Inspections/walkthroughs
  - Automation

E. Software Quality Assurance (SQA)

- Big picture – developing the right product at minimal cost
  - Making errors costs – fixing errors is wasted effort
  - The cost of fixing error increases with the time the error goes undetected
  - Both verification and validation applied *throughout the development process*
- SQA is concerned about software standards and processes
  - Standards define the artifacts of the software development/test process
  - Assumption: following the process leads to best chance of hitting the target
  - Product health is an indicator of how healthy the process is
  - Defects eventually are mapped back to the process
• Defect prevention/removal are process issues
• Leaders in SQA
  • Software Engineering Institute (SEI) – national (DOD) resource
  • Best practices
  • Training resources
• Barry Boehm – renowned software engineer management expert
  • Spiral model
  • COCOMO model
  • "Improving Software Productivity" classic paper
• Dr. Harlan Mills – cleanroom software engineering
  • Increased used of mathematics in specification and verification
  • Goal: zero-defect software development

F. Process Improvement
• Where to begin?
  • Gather data to find problem areas and associate costs
  • Improve areas with highest benefit-to-cost
• International Standards Organization (ISO 9000) models
• SEI Capability Maturity Model of organizational improvement