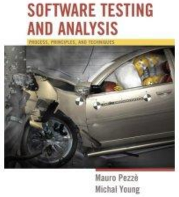
Unit 6
A Framework for Testing and Analysis

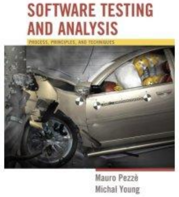
Mr. C. R. Belavi

Asst. Professor, HSIT, Nidasoshi



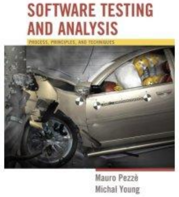
TOPICS IN UNIT 6

- **Process Framework:** Validation and verification, Degrees of freedom, Varieties of software. Basic principles: Sensitivity, redundancy, restriction, partition, visibility, Feedback. The quality process, Planning and monitoring Quality goals, Dependability properties, Analysis, Testing, Improving the process, Organizational factors.



contents

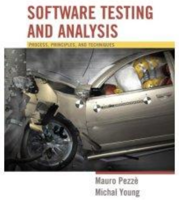
- Validation and verification
- Degree of freedom
- Verities of software
- Basic Definition



Why Software Testing

- To get good quality product.

- To find defects



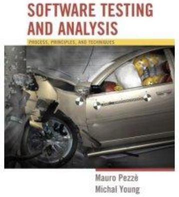
Verification and validation

- Validation:
  does the software system meets the user's real needs?

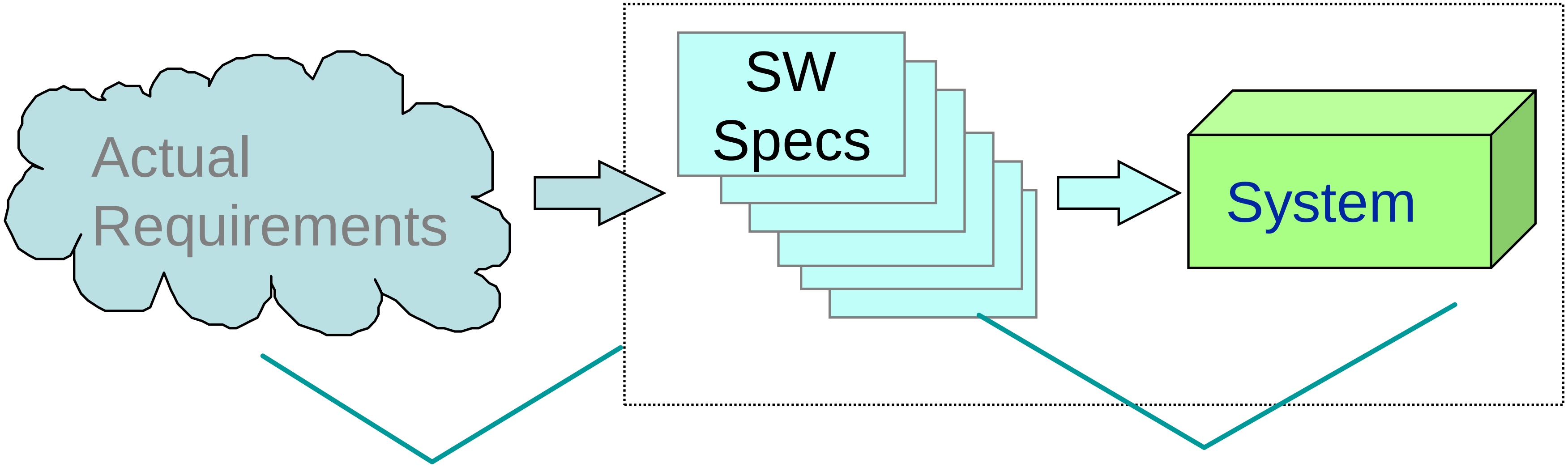
  are we building the right software?

- Verification:
  does the software system meets the requirements specifications?

  are we building the software right?



Validation and Verification

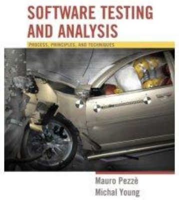


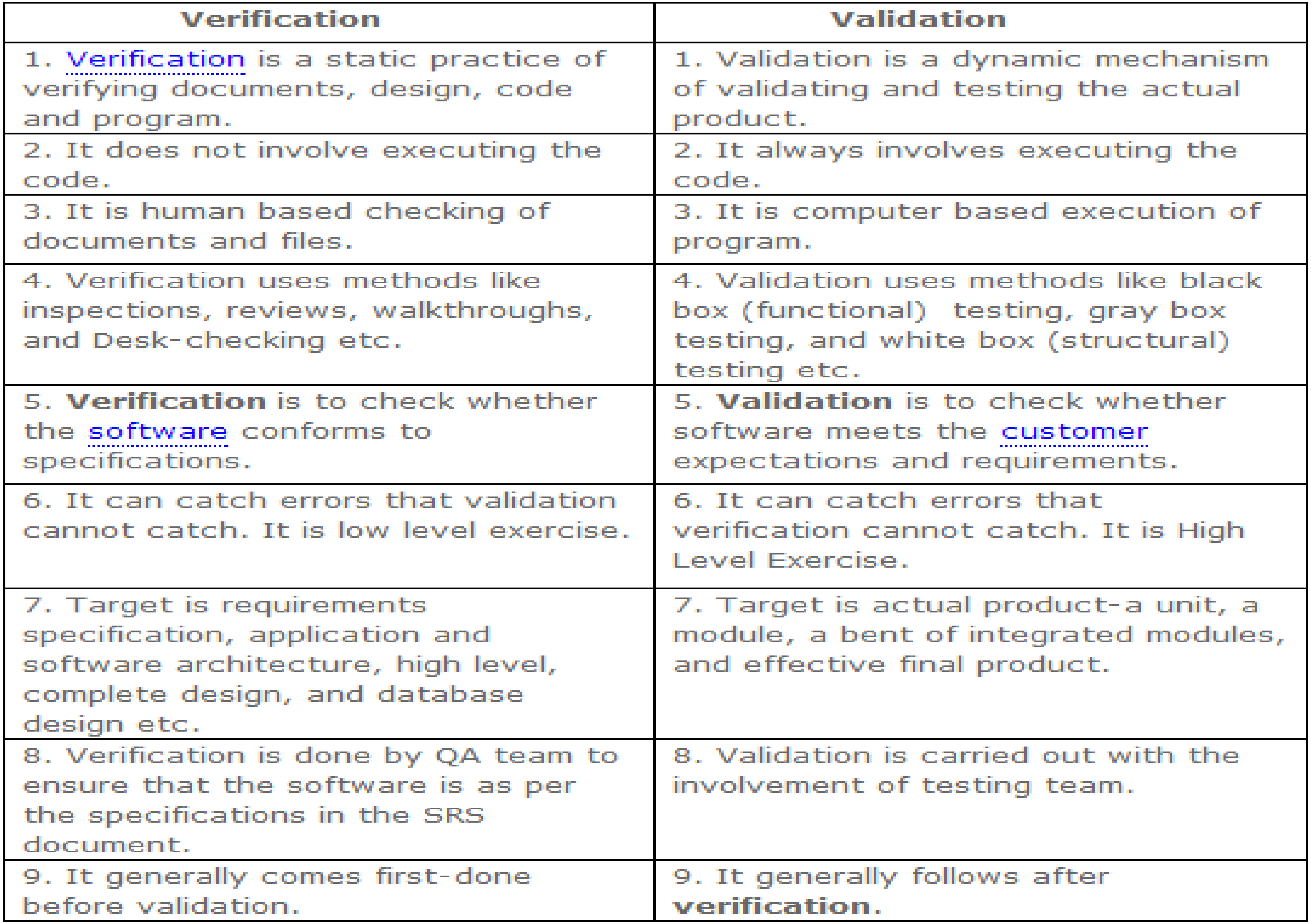
Validation

Includes usability testing, user feedback

Verification

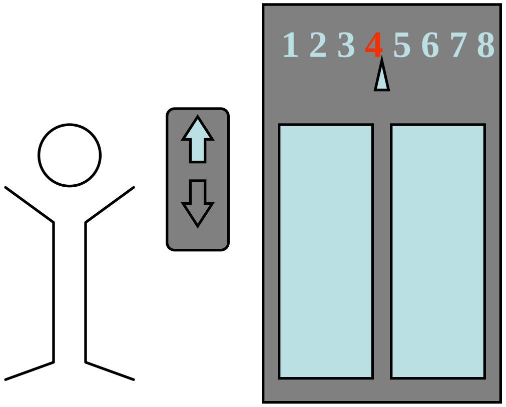
Includes testing, inspections, static analysis





Verification	Validation
1. Verification is a static practice of verifying documents, design, code and program.	1. Validation is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. Validation is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after **verification**.

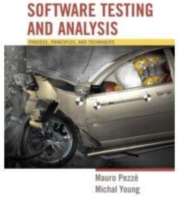
Verification or validation depends on the specification



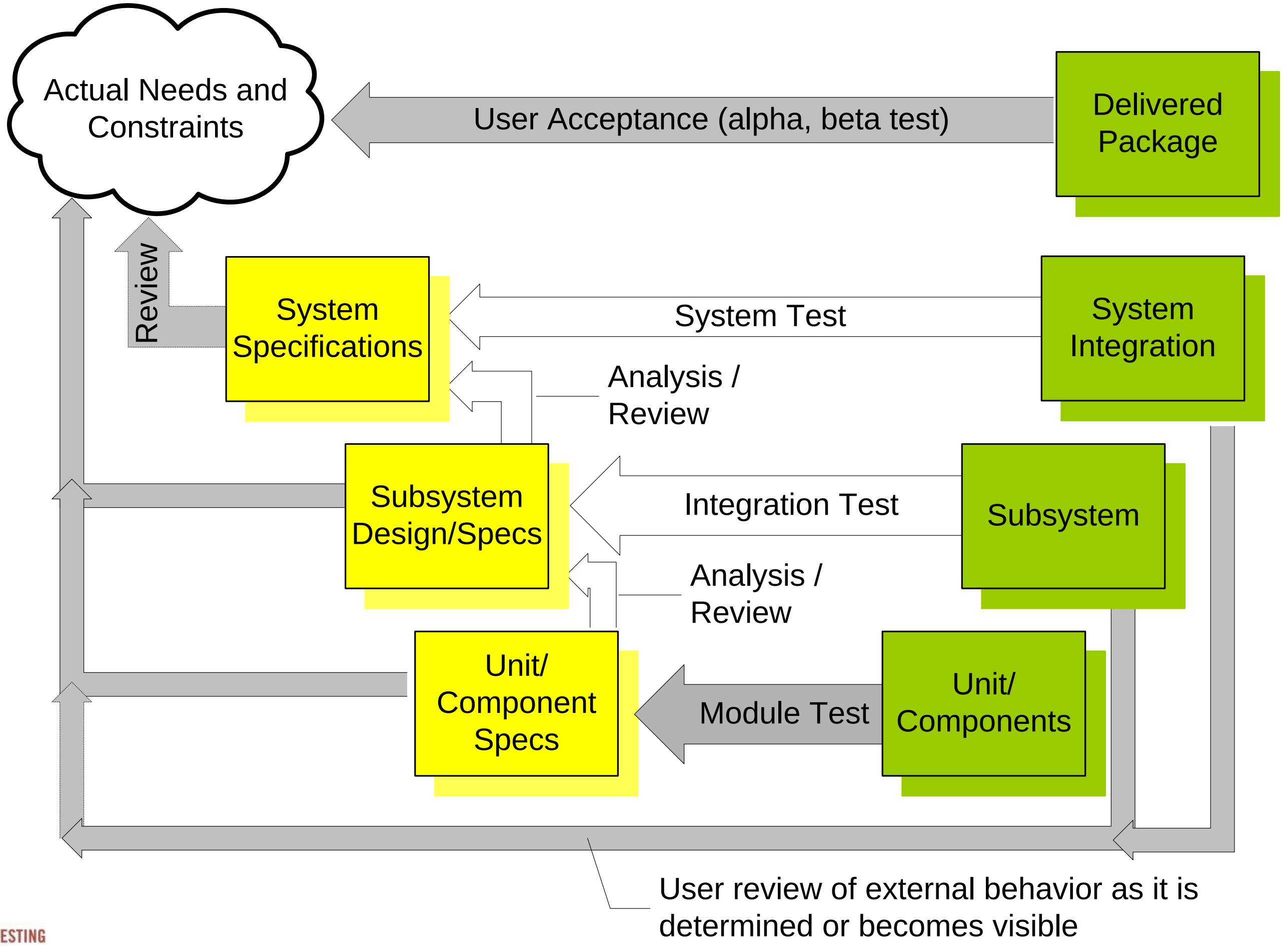
Example: elevator response

Unverifiable (but validatable) spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i soon...

Verifiable spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i within 30 seconds...


SOFTWARE TESTING AND ANALYSIS
PROCESS, PRINCIPLES, AND TECHNIQUES
Mauro Pezzè
Michal Young

Validation and Verification Activities



Actual Needs and Constraints

User Acceptance (alpha, beta test)

Delivered Package

Review

System Specifications

System Test

System Integration

Analysis / Review

Subsystem Design/Specs

Integration Test

Subsystem

Analysis / Review

Unit/ Component Specs

Module Test

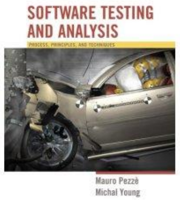
Unit/ Components

User review of external behavior as it is determined or becomes visible

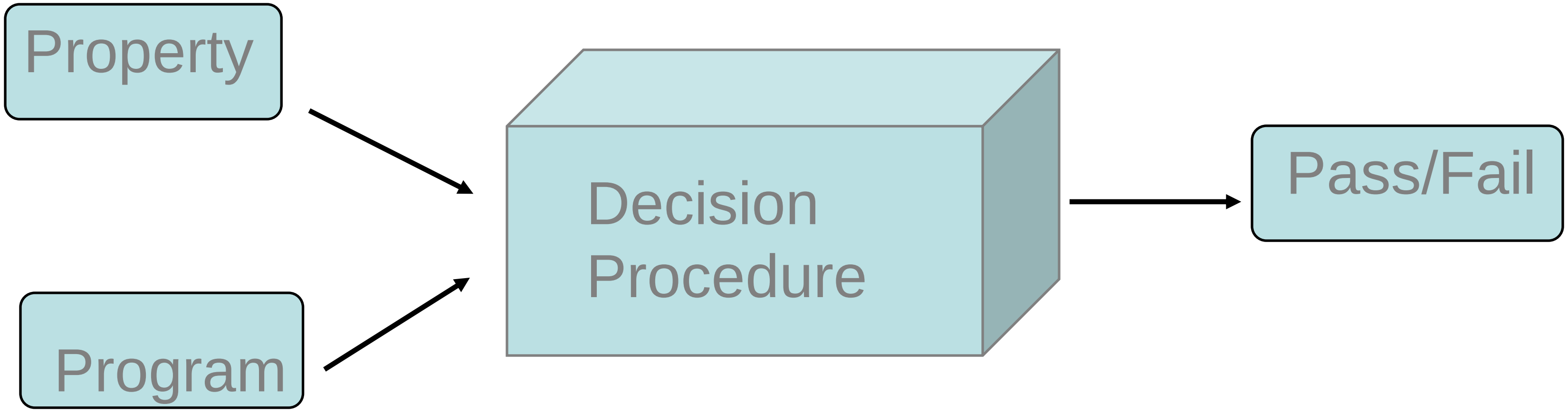
Sketches the relation of verification & validation activities With Respect To artifacts produced in a software development project.

validation

verification

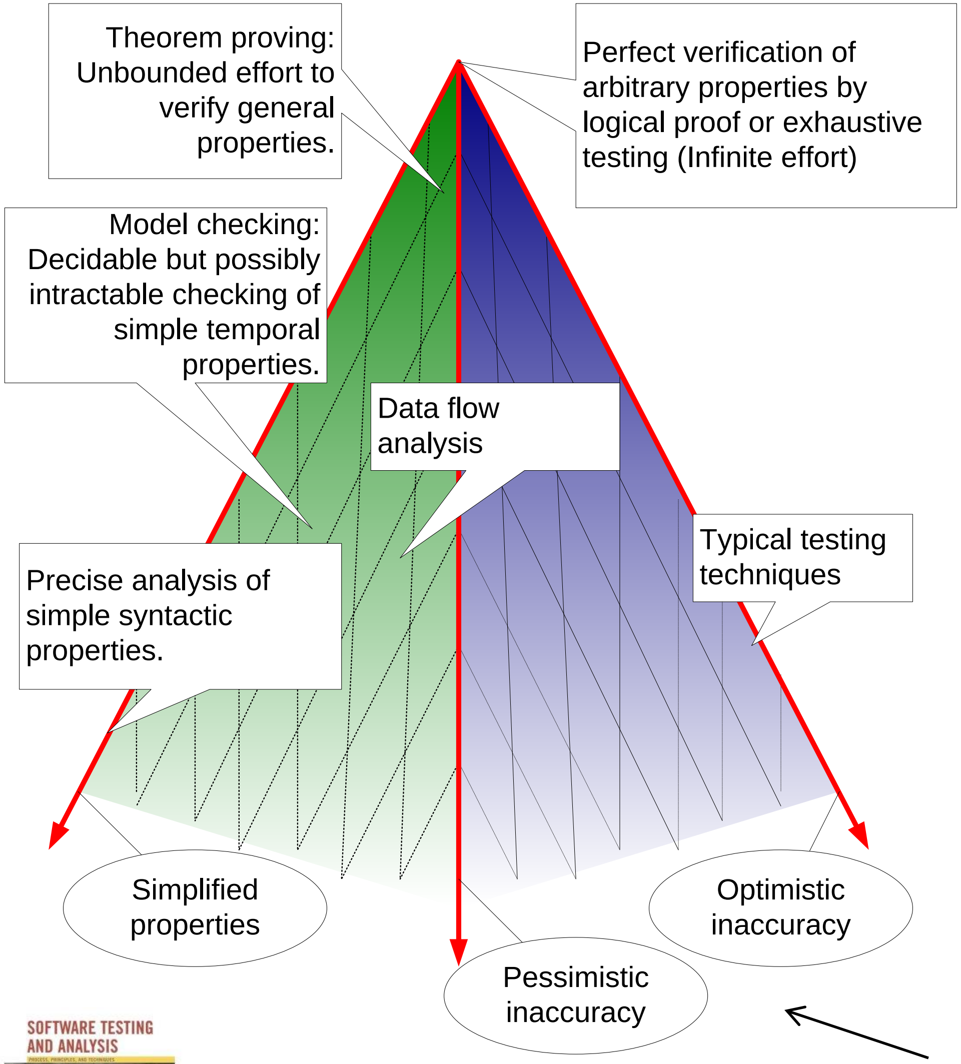


Degrees of freedom



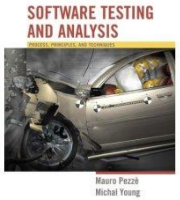
- Apply mathematical logic to verification of program.
- Alen Turing: some problems cannot be solved by any computer program.
- For most programs, exhaustive testing cannot be completed in any finite amount of time.
- You can’t always(ever) get what you want.
- Correctness properties are undecidable
- The halting problem can be embedded in almost every property of interest

Getting what you need ...


Theorem proving: Unbounded effort to verify general properties.
Perfect verification of arbitrary properties by logical proof or exhaustive testing (Infinite effort)
Model checking: Decidable but possibly intractable checking of simple temporal properties.
Data flow analysis
Typical testing techniques
Precise analysis of simple syntactic properties.
Simplified properties
Pessimistic inaccuracy
Optimistic inaccuracy

Verification trade-off dimensions

- A technique for verifying a property can be inaccurate in 1 of 2 directions.
- optimistic inaccuracy: we may accept some programs that do not possess the property (i.e., it may not detect all violations).
  - testing
- pessimistic inaccuracy: it is not guaranteed to accept a program even if the program does possess the property being analyzed
  - automated program analysis techniques
- simplified properties: reduce the degree of freedom for simplifying the property to check


SOFTWARE TESTING AND ANALYSIS
PROCESS, PRINCIPLES, AND TECHNIQUES
Mauro Pezzè
Michal Young

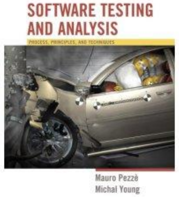
Dependability properties

1.Correctness

2.Reliability

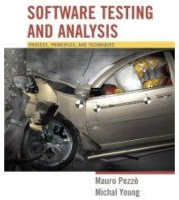
3. Robustness

4. safety



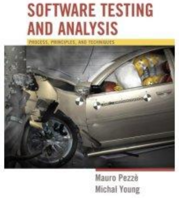
Some Terminology

- **Safe**: A safe analysis has no optimistic inaccuracy, i.e., it accepts only correct programs.
- **Sound**: An analysis of a program P with respect to a formula F is sound if the analysis returns true only when the program does satisfy the formula.
- **Complete**: An analysis of a program P with respect to a formula F is complete if the analysis always returns true when the program actually does satisfy the formula.



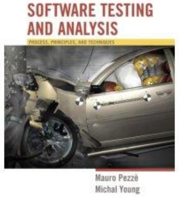
Basic Definition

- Sensitivity: Better to fail every time than sometimes.
- Redundancy: Error detection
- Restriction: unsolved problem into simple solution
- Partition: also known as Divide and conquer
- Visibility: measure the progress
- Feedback: it is process improvement.



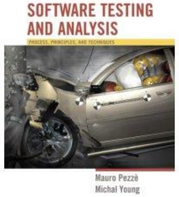
content

- Quality process
- Planning and monitoring
- Quality goals
- Dependability properties



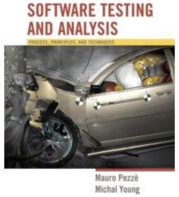
The software process

- A structured set of activities required to develop a software system
  - Specification;
  - Design;
  - Validation;
  - Evolution.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.



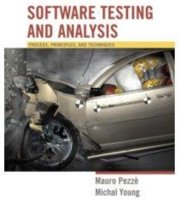
Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development.
- Evolutionary development
  - Specification, development and validation are interleaved.
- Component-based software engineering
  - The system is assembled from existing components.
- There are many variants of these models e.g. formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.



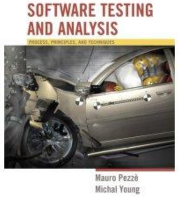
Software Qualities and Process

- Qualities cannot be added after development
  - Quality results from a set of inter-dependent activities
  - Analysis and testing are crucial but far from sufficient.
- Testing is not a phase, but a lifestyle
  - Testing and analysis activities occur from early in requirements engineering through delivery and subsequent evolution.
  - Quality depends on every part of the software process
- An essential feature of software processes is that software test and analysis is thoroughly integrated and not an afterthought



The Quality Process

- Quality process: set of activities and responsibilities
  - focused primarily on ensuring adequate dependability
  - concerned with project schedule or with product usability
- The quality process provides a framework for
  - selecting and arranging activities
  - considering interactions and trade-offs with other important goals.

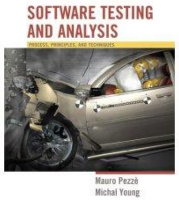


Interactions and tradeoffs

example

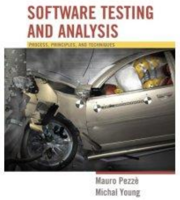
high dependability vs. time to market

- Mass market products:
  - better to achieve a reasonably high degree of dependability on a tight schedule than to achieve ultra-high dependability on a much longer schedule
- Critical medical devices:
  - better to achieve ultra-high dependability on a much longer schedule than a reasonably high degree of dependability on a tight schedule



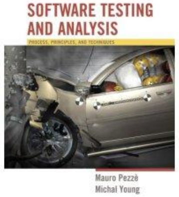
Properties of the Quality Process

- **Completeness**: Appropriate activities are planned to detect each important class of faults.
- **Timeliness**: Faults are detected at a point of high leverage (as early as possible)
- **Cost-effectiveness**: Activities are chosen depending on cost and effectiveness
  - cost must be considered over the whole development cycle and product life
  - the dominant factor is usually the cost of repeating an activity through many change cycles.



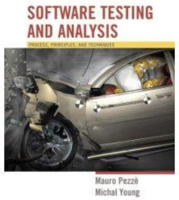
Planning and Monitoring

- The quality process
  - Balances several activities across the whole development process
  - Selects and arranges them to be as cost-effective as possible
  - Improves early visibility
- Quality goals can be achieved only through careful planning
- Planning is integral to the quality process



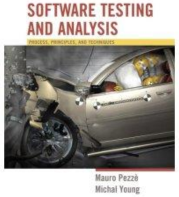
Process Visibility

- A process is visible to the extent that one can answer the question
  - How does our progress compare to our plan?
  - Example: Are we on schedule? How far ahead or behind?
- The quality process has not achieved adequate visibility if one cannot gain strong confidence in the quality of the software system before it reaches final testing
  - quality activities are usually placed as early as possible
    - design test cases at the earliest opportunity (not ``just in time")
    - uses analysis techniques on software artifacts produced before actual code.
  - motivates the use of “proxy” measures
    - Ex: the number of faults in design or code is not a true measure of reliability, but we may count faults discovered in design inspections as an early indicator of potential quality problems



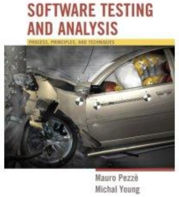
A&T Strategy

- Identifies company- or project-wide standards that must be satisfied
  - procedures required, e.g., for obtaining quality certificates
  - techniques and tools that must be used
  - documents that must be produced



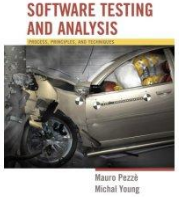
A&T Plan

- A comprehensive description of the quality process that includes:
  - objectives and scope of A&T activities
  - documents and other items that must be available
  - items to be tested
  - features to be tested and not to be tested
  - analysis and test activities
  - staff involved in A&T
  - constraints
  - pass and fail criteria
  - schedule
  - deliverables
  - hardware and software requirements
  - risks and contingencies



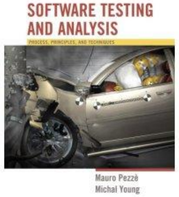
Quality Goals

- Process qualities (visibility,....)
- Product qualities
  - internal qualities (maintainability,....)
  - external qualities
    - usefulness qualities:
      - usability, performance, security, portability, interoperability
    - dependability
      - correctness, reliability, safety, robustness

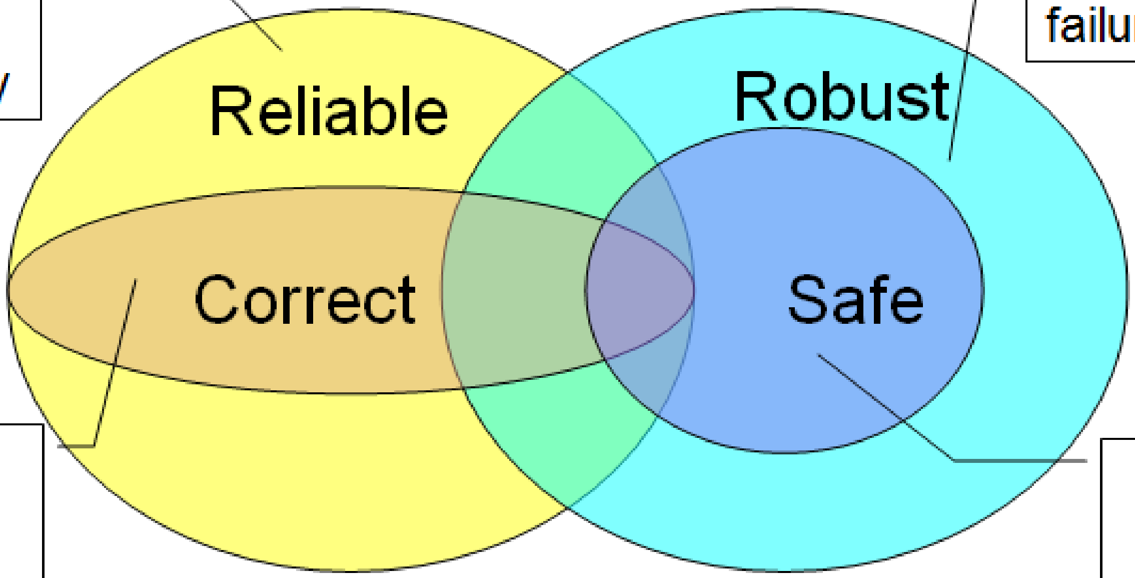


Dependability Qualities

- Correctness:
  - A program is correct if it is consistent with its specification
    - seldom practical for non-trivial systems
- Reliability:
  - likelihood of correct function for some ``unit" of behavior
    - relative to a specification and usage profile
    - statistical approximation to correctness (100% reliable = correct)
- Safety:
  - preventing hazards
- Robustness
  - acceptable (degraded) behavior under extreme conditions



Relation among Dependability Qualites

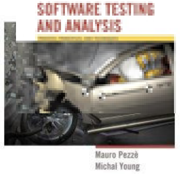


reliable but not correct: failures occur rarely

robust but not safe: catastrophic failures can occur

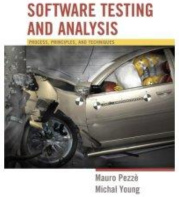
correct but not safe or robust: the specification is inadequate

safe but not correct: annoying failures can occur



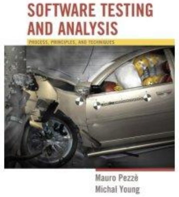
contents

- Analysis
- Testing
- Improving the process
- Organizational factors



Analysis

- analysis includes
  - manual inspection techniques
  - automated analyses
- can be applied at any development stage
- particularly well suited at the early stages of specifications an design



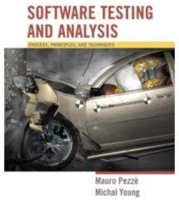
Inspection

- can be applied to essentially any document
  - requirements statements
  - architectural and detailed design documents
  - test plans and test cases
  - program source code

Drawbacks

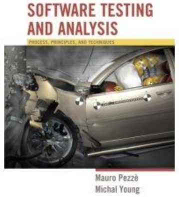
- takes a considerable amount of time and require meeting.
- re-inspecting a changed component can be expensive

- used primarily
  - where other techniques are inapplicable
  - where other techniques do not provide sufficient coverage



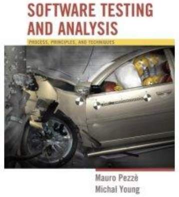
Software Inspections

- People examine a source code representation to discover anomalies and defects
- Does not require systems execution so they may occur before implementation
- May be applied to any system representation (document, model, test data, code, etc.)



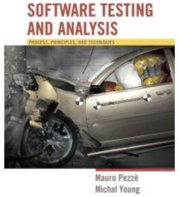
Inspection Preconditions

- A precise specification must be available
- Team members must be familiar with organization standards
- All representations must be syntactically correct
- An error checklist must be prepare in advance
- Management must buy into the the fact the inspections will increase the early development costs
- Inspections cannot be used to evaluate staff performance



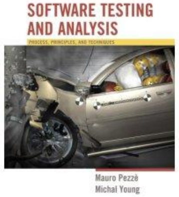
Inspection Procedure

- System overview presented to inspection team
- Code and associated documents are distributed to team in advance
- Errors discovered during the inspection are recorded
- Product modifications are made to repair defects
- Re-inspection may or may not be required



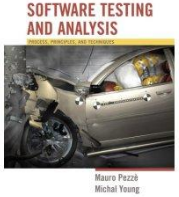
Inspection Teams

- Have at least 4 team members
  - product author
  - inspector (looks for errors, omissions, and inconsistencies)
  - reader (reads the code to the team)
  - moderator (chairs meeting and records errors uncovered)



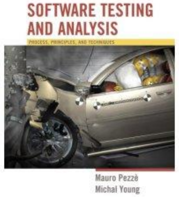
Inspection Checklists

- Checklists of common errors should be used to drive the inspection
- Error checklist should be language dependent
- The weaker the type checking in the language, the larger the checklist is likely to become



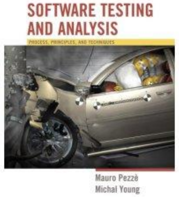
Inspection Fault Classes

- Data faults (e.g. array bounds)
- Control faults (e.g. loop termination)
- Input/output faults (e.g. all data read)
- Interface faults (e.g. parameter assignment)
- Storage management faults (e.g. memory leaks)
- Exception management faults (e.g. all error conditions trapped)



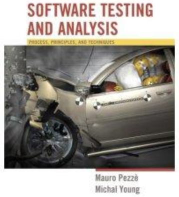
Inspection Rate

- 500 statements per hour during overview
- 125 statements per hour during individual preparation
- 90-125 statements per hour can be inspected by a team
- Including preparation time, each 100 lines of code costs one person day (if a 4 person team is used)



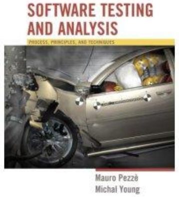
Automated Static analysis

- More limited in applicability
  - can be applied to some formal representations of requirements models
  - not to natural language documents
- are selected when available
  - substituting machine cycles for human effort makes them particularly cost-effective.



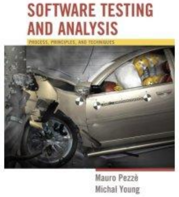
Testing

- Test are executed when the corresponding code is available, But the testing activities start earlier as soon as the artifacts are available.



Advantages of Testing

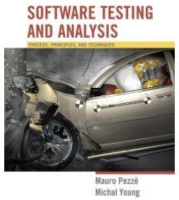
- Test case highlight inconsistencies and incompleteness in software specification.
- Test case allow for early repair of software specification. Preventing fault from propagating to later stages in development
- It clarify the specification for error and unexpected condition.
- “EARLY IS BETTER” Rule.



Improving the process

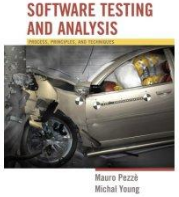
- To find cost effective counter measures for fault that are expensive because of

1. Occurs frequently
2. Failure that cause are expensive.
3. Expensive to repair.



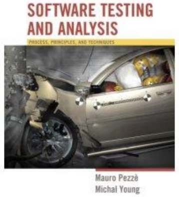
- Raw data on faults are gathered.
- Aggregated into categories based on cause
- Remedies.

Root cause Analysis.



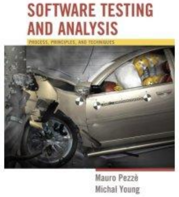
Organizational Factors

- Different teams for development and quality?
  - separate development and quality teams is common in large organizations
  - indistinguishable roles is postulated by some methodologies (extreme programming)
- Different roles for development and quality?
  - test designer is a specific role in many organizations
  - mobility of people and roles by rotating engineers over development and testing tasks among different projects is a possible option



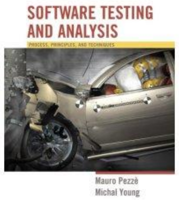
Example of Allocation of Responsibilities

- Allocating tasks and responsibilites is a complex job:
  we can allocate
  - Unit testing
    - to the development team (requires detailed knowledge of the code)
    - but the quality team may control the results (structural coverage)
  - Integration, system and acceptance testing
    - to the quality team
    - but the development team may produce scaffolding and oracles
  - Inspection and walk-through
    - to mixed teams
  - Regression testing
    - to quality and maintenance teams
  - Process improvement related activities
    - to external specialists interacting with all teams



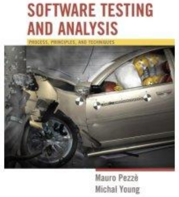
Allocation of Responsibilities and rewarding mechanisms: case A

- allocation of responsibilities
  - Development team responsible development m easured with LOC per person month
  - Quality team responsible for quality
- possible effect
  - Development team tries to maximize productivity, without considering quality
  - Quality team will not have enough resources for bad quality products
- **result**
  - **product of bad quality and overall project failure**



Allocation of Responsibilities and rewarding mechanisms: case B

- allocation of responsibilities
  - Development team responsible for both development and quality control
- possible effect
  - the problem of case A is solved
  - but the team may delay testing for development without leaving enough resources for testing
- result
  - delivery of a not fully tested product and overall project failure



summary

- A software does not fall neatly into one category but rather has a no of relevent characteristics, that must be considered when planning verification.
- Most interesting properties are undecidable, thus in general we cannot count on tools that work without human intevention
- Assessing program qualities comprises two complementary sets of activities: validation (daes the software do what it is supposed to do?) and verification (does the system behave as specificed?)
- There is no single technique for all purposes: test designers need to select a suitable combination of techniques