Introduction to software testing and quality process

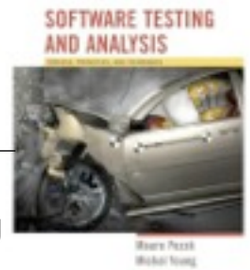
Automated testing and verification

J.P. Galeotti - Alessandra Gorla

Engineering processes

- Engineering disciplines pair
  - construction activities
  - activities that check intermediate and final products
- Software engineering is no exception: construction of high quality software requires
  - construction and
  - verification activities

(c) 2007 Mauro Pezzè & Michal Young



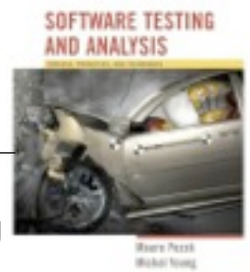
Peculiarities of software

- Software has some characteristics that make V&V particularly difficult:
    - Many different quality requirements
    - Evolving (and deteriorating) structure
    - Inherent non-linearity
    - Uneven distribution of faults

Example

- If an elevator can safely carry a load of 1000 kg, it can also safely carry any smaller load; If a procedure correctly sorts a set of 256 elements, it may fail on a set of 255 or 53 or 12 elements, as well as on 257 or 1023.

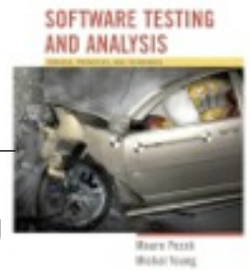
(c) 2007 Mauro Pezzè & Michal Young



Impact of new technologies

- Advanced development technologies
  - can reduce the frequency of some classes of errors
  - but do not eliminate errors
- New development approaches can introduce new kinds of faults
  - Memory management improved in java in respect to c
  - new problems due to the use of polymorphism, dynamic binding and private state in object-oriented software.

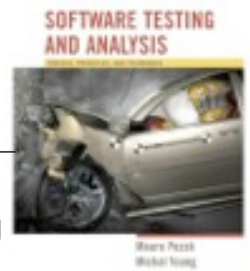
(c) 2007 Mauro Pezzè & Michal Young



Variety of approaches

- There are no fixed recipes
- Quality managers must
  - choose and schedule the right blend of techniques
    - to reach the required level of quality
    - within cost constraints
- design a specific solution that suits
  - the problem
  - the requirements
  - the development environment

(c) 2007 Mauro Pezzè & Michal Young



Five Basic Questions

- When do verification and validation start?
  When are they complete?

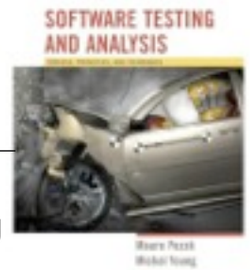
- What particular techniques should be applied during development?

- How can we assess the readiness of a product?

- How can we control the quality of successive releases?

- How can the development process itself be improved?

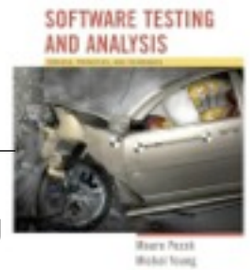
(c) 2007 Mauro Pezzè & Michal Young



1: When do V&V start? When are they complete?

- Test is not a (late) phase of software development
  - Execution of tests is a small part of the verification and validation process
- V&V start as soon as we decide to build a software product, or even before
- V&V last far beyond the product delivery as long as the software is in use, to cope with evolution and adaptations to new conditions

(c) 2007 Mauro Pezzè & Michal Young



Early start: from feasibility study

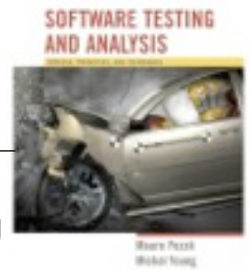
- The feasibility study of a new project must take into account the required qualities and their impact on the overall cost
- At this stage, quality related activities include
  - risk analysis
  - measures needed to assess and control quality at each stage of development.
  - assessment of the impact of new features and new quality requirements
  - contribution of quality control activities to development cost and schedule.

(c) 2007 Mauro Pezzè & Michal Young

Long lasting: beyond maintenance

- Maintenance activities include
  - analysis of changes and extensions
  - generation of new test suites for the added functionalities
  - re-executions of tests to check for non regression of software functionalities after changes and extensions
  - fault tracking and analysis

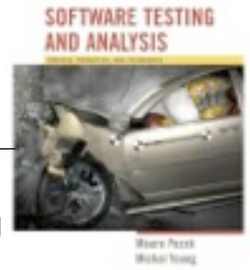
(c) 2007 Mauro Pezzè & Michal Young

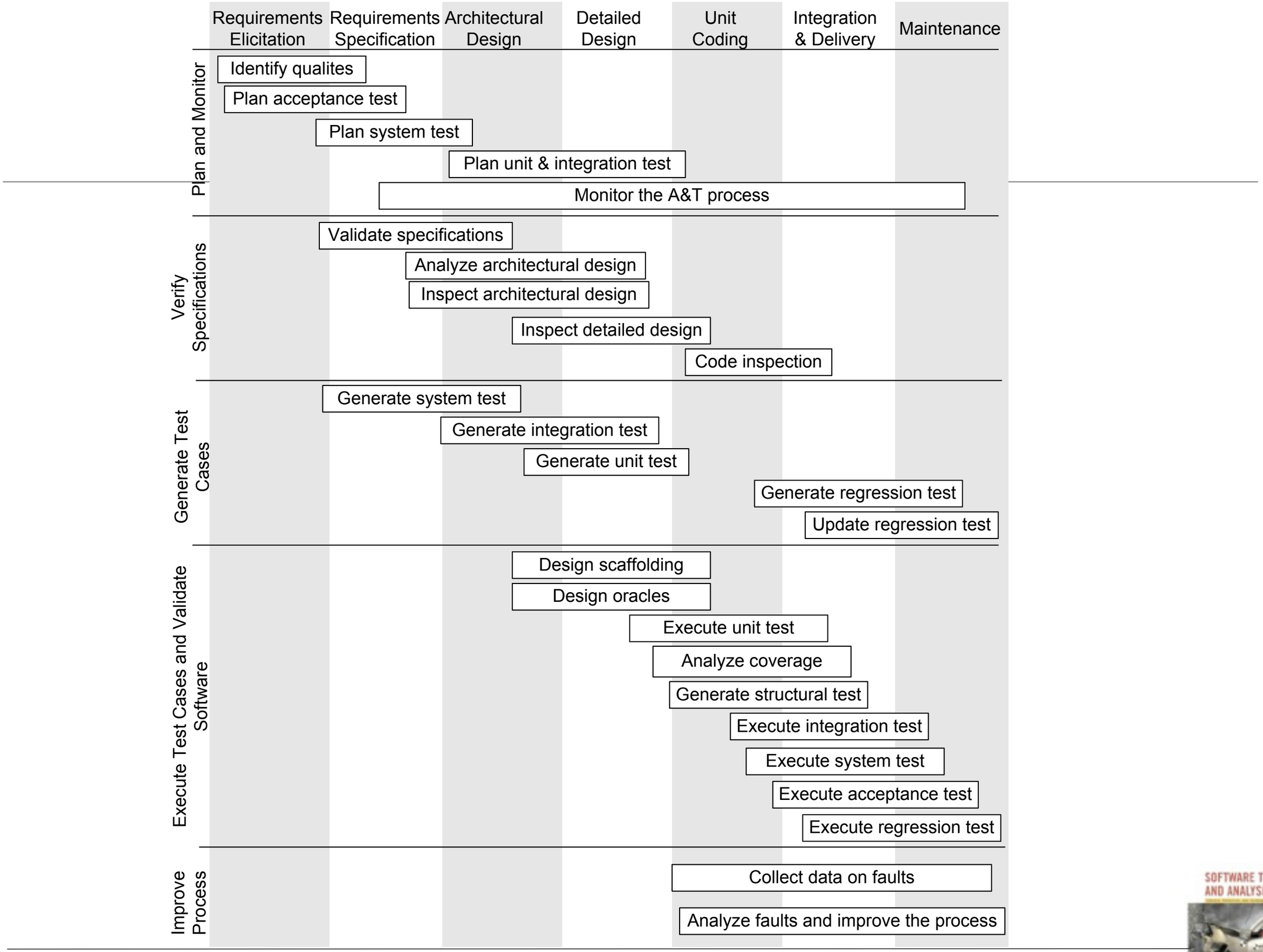


2: What particular techniques should be applied during development?

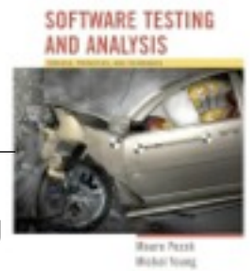
- No single A&T technique can serve all purposes
- The primary reasons for combining techniques are:
  - Effectiveness for different classes of faults
    example: analysis instead of testing for race conditions
  - Applicability at different points in a project
    example: inspection for early requirements validation
  - Differences in purpose
    example: statistical testing to measure reliability
  - Tradeoffs in cost and assurance
    example: expensive technique for key properties

(c) 2007 Mauro Pezzè & Michal Young





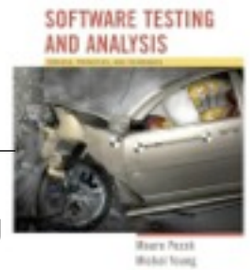
(c) 2007 Mauro Pezzè & Michal Young



3: How can we assess the readiness of a product?

- A&T during development aim at revealing faults
- We cannot reveal and remove all faults
- A&T cannot last indefinitely: we want to know if products meet the quality requirements
- We must specify the required level of dependability
- and determine when that level has been attained.

(c) 2007 Mauro Pezzè & Michal Young

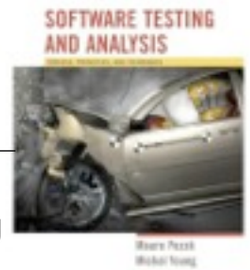


Different measures of dependability

- Availability measures the quality of service in terms of running versus down time

- Mean time between failures (MTBF) measures the quality of the service in terms of time between failures

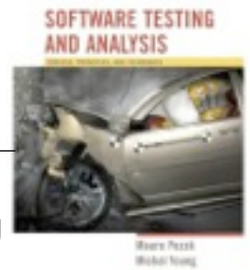
- Reliability indicates the fraction of all attempted operations that complete successfully



Assessing dependability

- Randomly generated tests following an operational profile
- Alpha test: tests performed by users in a controlled environment, observed by the development organization
- Beta test: tests performed by real users in their own environment, performing actual tasks without interference or close monitoring

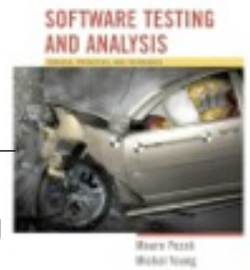
(c) 2007 Mauro Pezzè & Michal Young



4: How can we control the quality of successive releases?

- Software test and analysis does not stop at the first release.
- Software products operate for many years, and undergo many changes:
  - They adapt to environment changes
  - They evolve to serve new and changing user requirements.
- Quality tasks after delivery
  - test and analysis of new and modified code
  - re-execution of system tests
  - extensive record-keeping

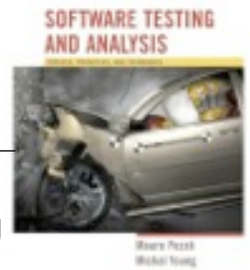
(c) 2007 Mauro Pezzè & Michal Young



5: How can the development process itself be improved?

- The same defects are encountered in project after project
- A third goal of the improving the quality process is to improve the process by
  - identifying and removing weaknesses in the development process
  - identifying and removing weaknesses in test and analysis that allow them to remain undetected

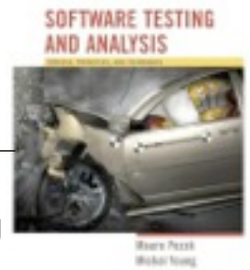
(c) 2007 Mauro Pezzè & Michal Young



A four step process to improve fault analysis and process

- Define the data to be collected and implement procedures for collecting them
- Analyze collected data to identify important fault classes
- Analyze selected fault classes to identify weaknesses in development and quality measures
- Adjust the quality and development process

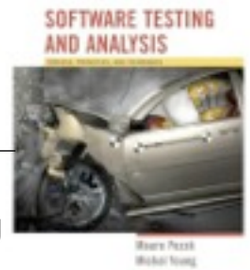
(c) 2007 Mauro Pezzè & Michal Young



Summary

- The quality process has three different goals:
  - Improving a software product
  - assessing the quality of the software product
  - improving the quality process
- We need to combine several A&T techniques through the software process
- A&T depend on organization and application domain.
- Cost-effectiveness depends on the extent to which techniques can be re-applied as the product evolves.
- Planning and monitoring are essential to evaluate and refine the quality process.

(c) 2007 Mauro Pezzè & Michal Young



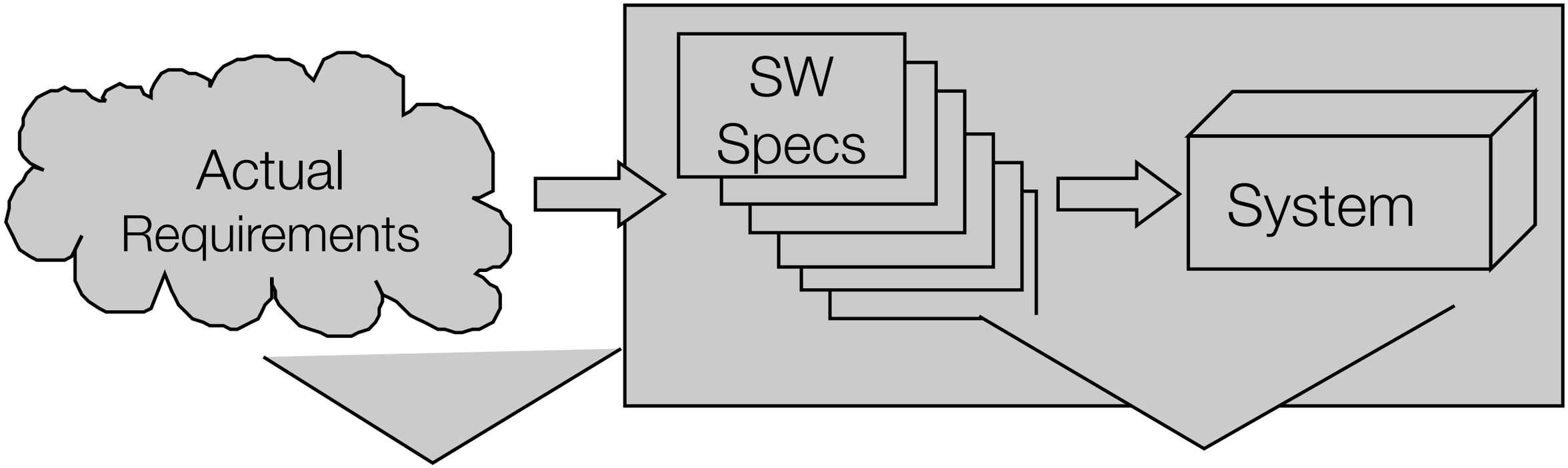
Dimensions and tradeoffs of testing and analysis techniques

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?

(c) 2007 Mauro Pezzè & Michal Young

Validation and Verification

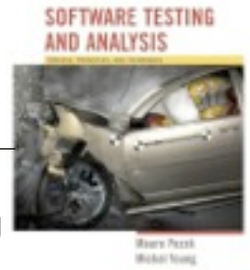


Validation

Includes usability testing, user feedback

Verification

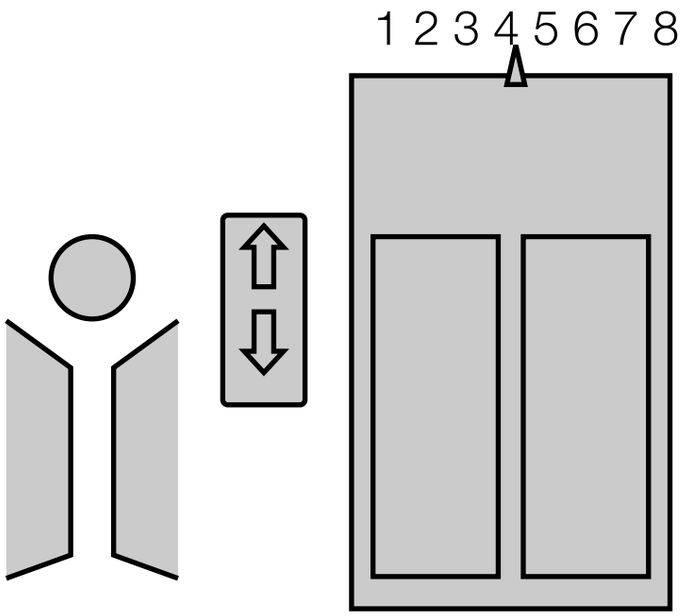
Includes testing, inspections, static analysis



(c) 2007 Mauro Pezzè & Michal Young

Verification or validation depends on the specification

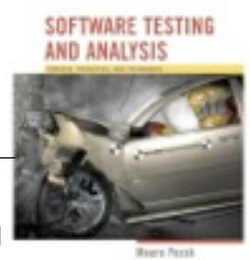
- Unverifiable (but validatable) spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i *soon*...



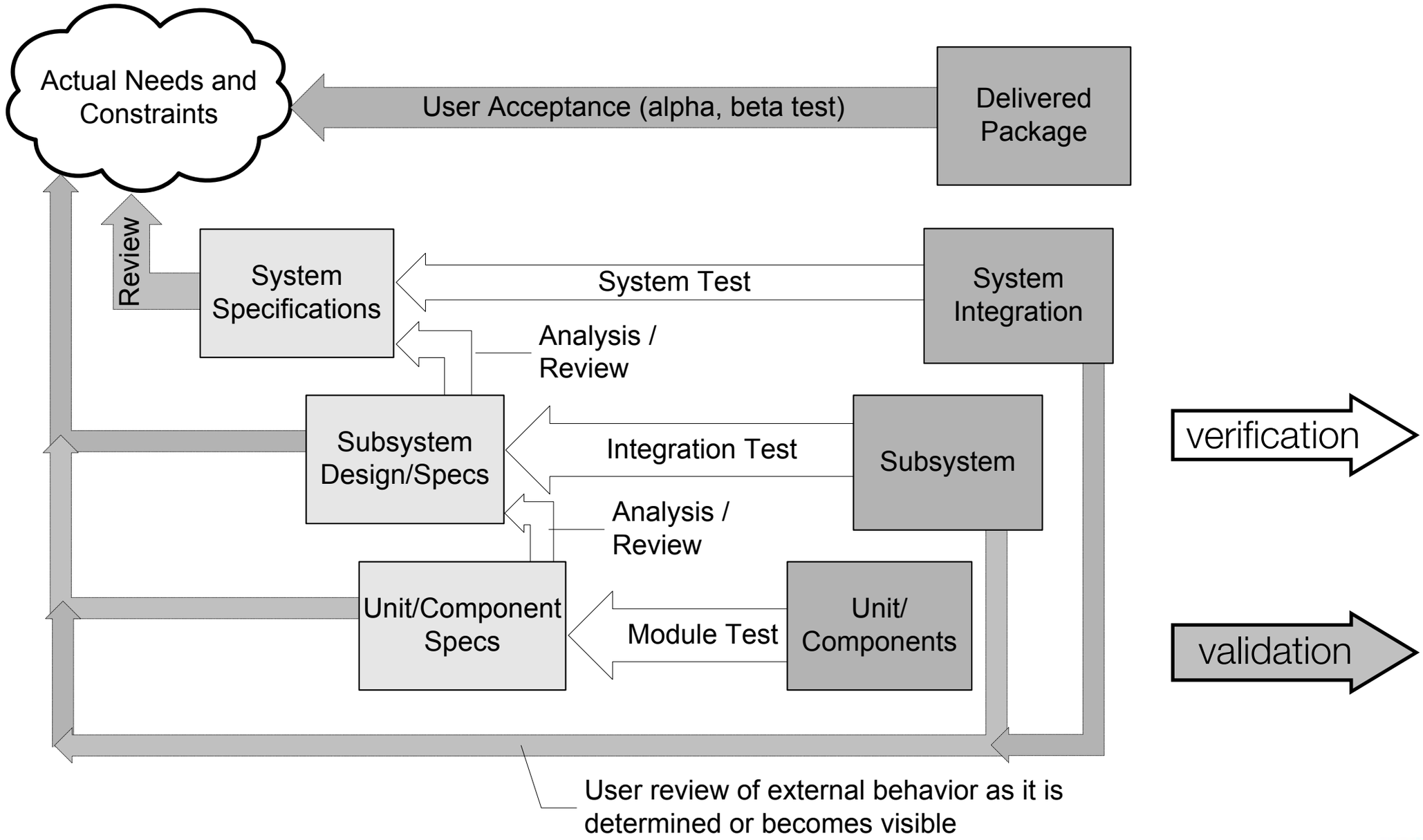
Example: elevator response

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

(c) 2007 Mauro Pezzè & Michal Young



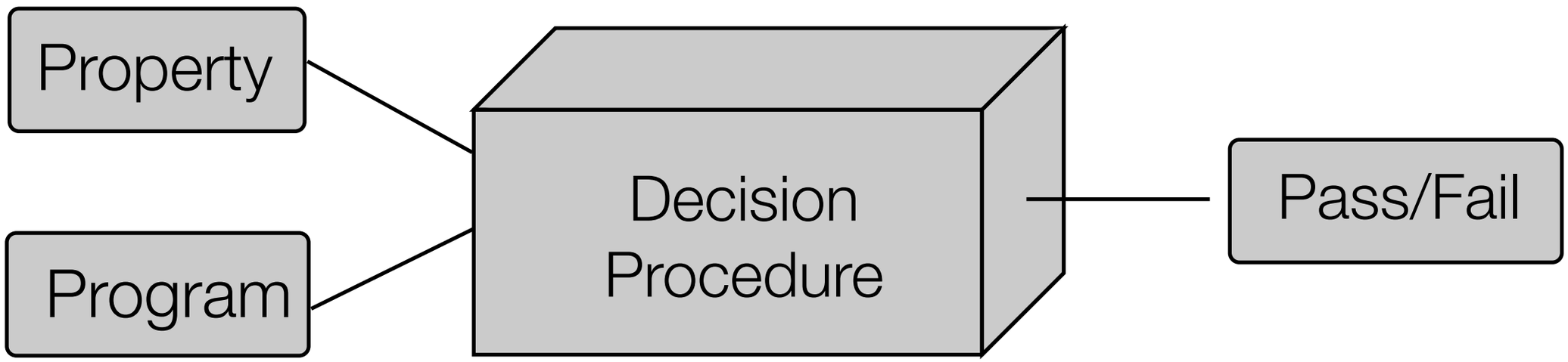
Validation and Verification Activities



(c) 2007 Mauro Pezzè & Michal Young

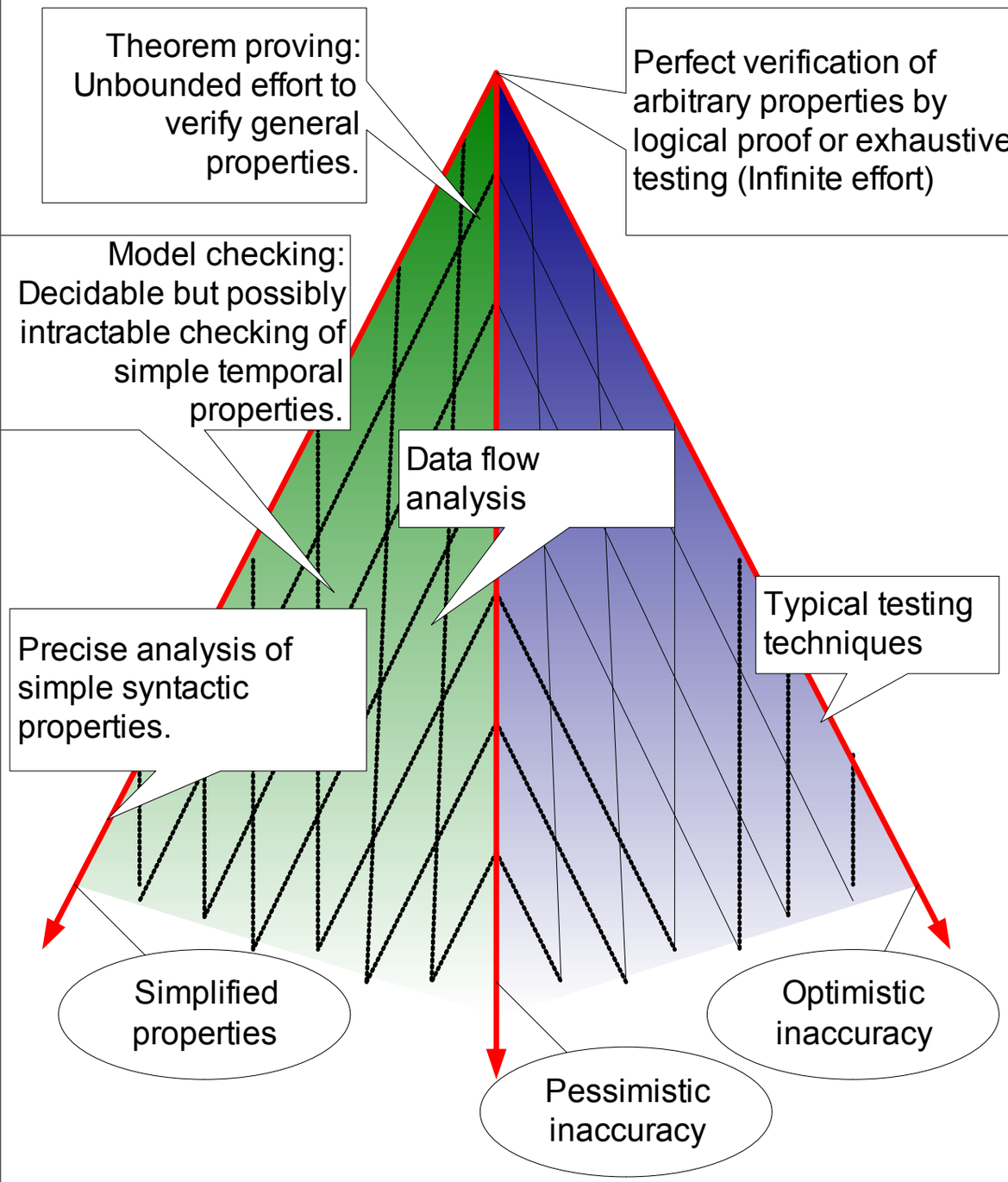
You can’t always ~~get~~ ever what you want

- Correctness properties are undecidable
  - the halting problem can be embedded in almost every property of interest



(c) 2007 Mauro Pezzè & Michal Young

Getting what you need ...



- **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

(c) 2007 Mauro Pezzè & Michal Young

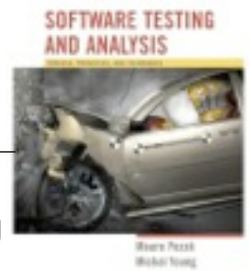
Summary

- Most interesting properties are undecidable, thus in general we cannot count on tools that work without human intervention

- Assessing program qualities comprises two complementary sets of activities: validation (does the software do what it is supposed to do?) and verification (does the system behave as specified?)

- There is no single technique for all purposes: test designers need to select a suitable combination of techniques

(c) 2007 Mauro Pezzè & Michal Young

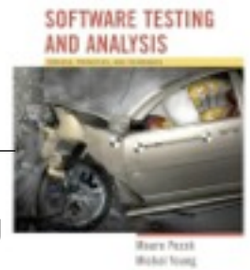


Some important principles

Main A&T Principles

- General engineering principles:
  - *Partition*: divide and conquer
  - *Visibility*: making information accessible
  - *Feedback*: tuning the development process
- Specific A&T principles:
  - *Sensitivity*: better to fail every time than sometimes
  - *Redundancy*: making intentions explicit
  - *Restriction*: making the problem easier

(c) 2007 Mauro Pezzè & Michal Young



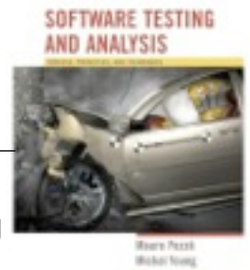
Sensitivity: better to fail every time than sometimes

- Consistency helps:
  - a test selection criterion works better if every selected test provides the same result, i.e., if the program fails with one of the selected tests, it fails with all of them (reliable criteria)
  - run time deadlock analysis works better if it is machine independent, i.e., if the program deadlocks when analyzed on one machine, it deadlocks on every machine

Redundancy: making intentions explicit

- Redundant checks can increase the capabilities of catching specific faults early or more efficiently.
    - Static type checking is redundant with respect to dynamic type checking, but it can reveal many type mismatches earlier and more efficiently.
- Validation of requirement specifications is redundant with respect to validation of the final software, but can reveal errors earlier and more efficiently.
- Testing and proof of properties are redundant, but are often used together to increase confidence

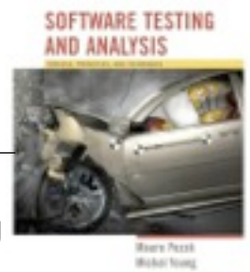
(c) 2007 Mauro Pezzè & Michal Young



Partition: divide and conquer

- Hard testing and verification problems can be handled by suitably partitioning the input space:
  - both structural and functional test selection criteria identify suitable partitions of code or specifications (partitions drive the sampling of the input space)
  - verification techniques fold the input space according to specific characteristics, grouping homogeneous data together and determining partitions

(c) 2007 Mauro Pezzè & Michal Young



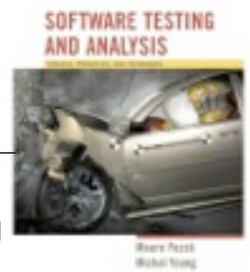
Restriction: making the problem easier

- Suitable restrictions can reduce hard (unsolvable) problems to simpler (solvable) problems

  - A weaker spec may be easier to check: it is impossible (in general) to show that pointers are used correctly, but the simple Java requirement that pointers are initialized before use is simple to enforce.

  - A stronger spec may be easier to check: it is impossible (in general) to show that type errors do not occur at run-time in a dynamically typed language, but statically typed languages impose stronger restrictions that are easily checkable.

(c) 2007 Mauro Pezzè & Michal Young



Visibility: Judging status

- The ability to measure progress or status against goals
    - X visibility = ability to judge how we are doing on X, e.g., schedule visibility = “Are we ahead or behind schedule,” quality visibility = “Does quality meet our objectives?”
    - Involves setting goals that can be assessed at each stage of development
        - The biggest challenge is early assessment, e.g., assessing specifications and design with respect to product quality
- Related to observability
    - Example: Choosing a simple or standard internal data format to facilitate unit testing

(c) 2007 Mauro Pezzè & Michal Young

Feedback: tuning the development process

- Learning from experience: Each project provides information to improve the next

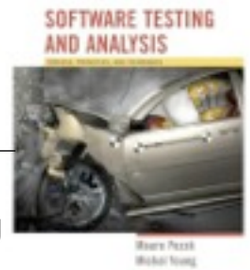
- Examples

  - Checklists are built on the basis of errors revealed in the past

  - Error taxonomies can help in building better test selection criteria

  - Design guidelines can avoid common pitfalls

(c) 2007 Mauro Pezzè & Michal Young



Summary

- The discipline of test and analysis is characterized by 6 main principles:
  - Sensitivity: better to fail every time than sometimes
  - Redundancy: making intentions explicit
  - Restriction: making the problem easier
  - Partition: divide and conquer
  - Visibility: making information accessible
  - Feedback: tuning the development process
- They can be used to understand advantages and limits of different approaches and compare different techniques

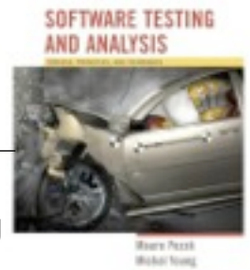
(c) 2007 Mauro Pezzè & Michal Young

Test and Analysis Activities within a Software Process

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

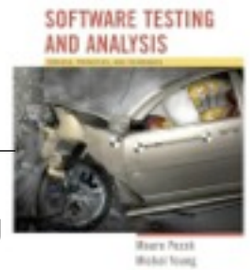
(c) 2007 Mauro Pezzè & Michal Young



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.

(c) 2007 Mauro Pezzè & Michal Young



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

(c) 2007 Mauro Pezzè & Michal Young

Properties of the Quality Process

- *Completeness*: Appropriate activities are planned to detect each important class of faults.

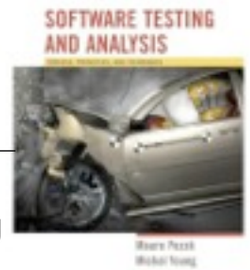
- *Timeliness*: Faults are detected at a point of high leverage (as early as possible)

- *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.

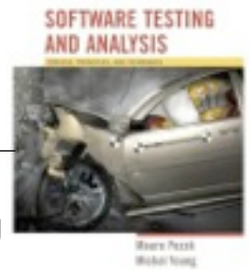
(c) 2007 Mauro Pezzè & Michal Young



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

(c) 2007 Mauro Pezzè & Michal Young



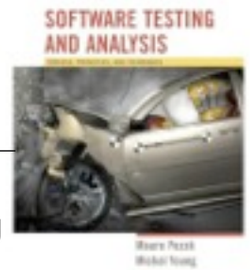
Process Visibility

- 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

(c) 2007 Mauro Pezzè & Michal Young

A&T Strategy

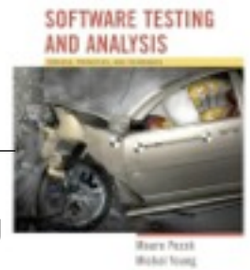
- Identifies company- or project-wide standards that must be satisfied
  - procedures required, e.g., for obtaining quality certificates
  - techniques and tools that must be used
  - documents that must be produced



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

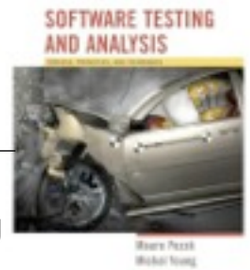
(c) 2007 Mauro Pezzè & Michal Young



Quality Goals

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

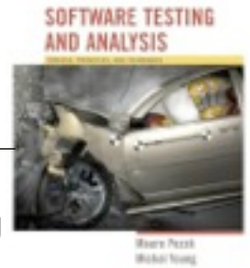
(c) 2007 Mauro Pezzè & Michal Young



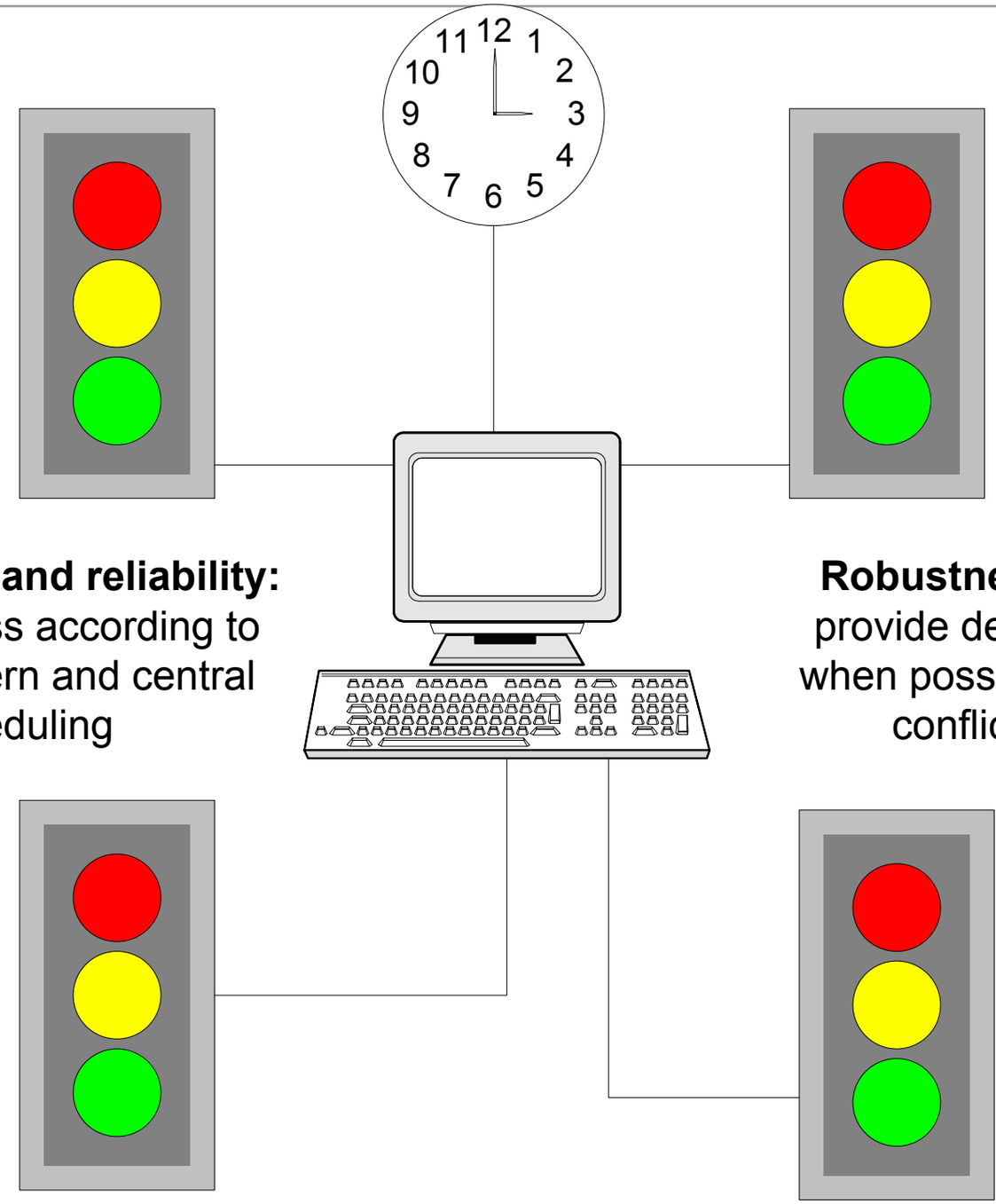
Dependability Qualities

- 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

(c) 2007 Mauro Pezzè & Michal Young



Example of Dependability Qualities

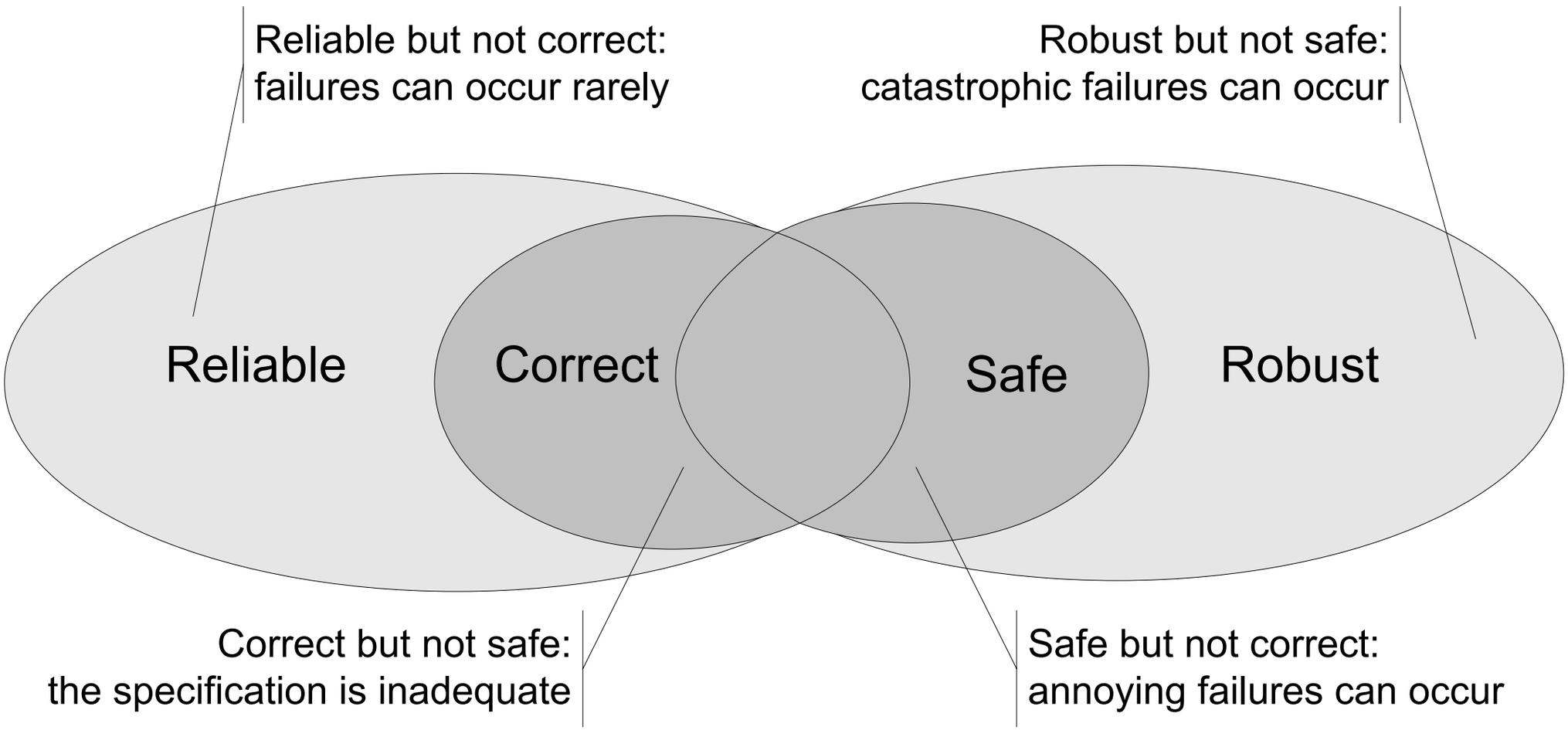


Correctness and reliability: let traffic pass according to correct pattern and central scheduling

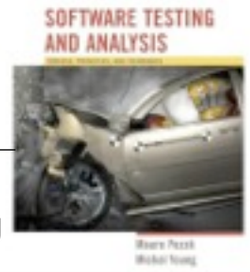
Robustness and safety: provide degraded function when possible; never signal conflicting greens

(c) 2007 Mauro Pezzè & Michal Young

Relation among Dependability Qualities



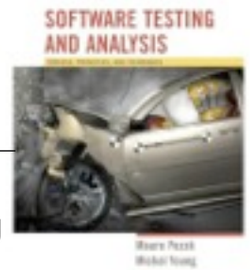
(c) 2007 Mauro Pezzè & Michal Young



Analysis

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

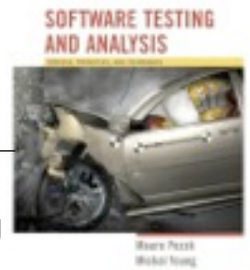
(c) 2007 Mauro Pezzè & Michal Young



Inspection

- can be applied to essentially any document
  - requirements statements
  - architectural and detailed design documents
  - test plans and test cases
  - program source code
- may also have secondary benefits
  - spreading good practices
  - instilling shared standards of quality.
- takes a considerable amount of time
- re-inspecting a changed component can be expensive
- used primarily
  - where other techniques are inapplicable
  - where other techniques do not provide sufficient coverage

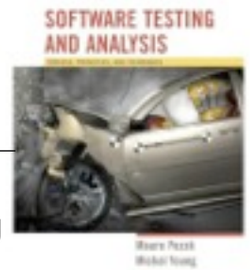
(c) 2007 Mauro Pezzè & Michal Young



Automatic 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.

(c) 2007 Mauro Pezzè & Michal Young



Testing

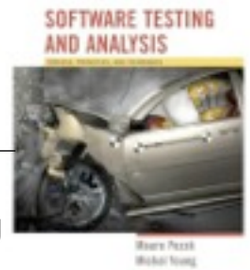
- Executed late in development
- Start as early as possible
- Early test generation has several advantages
  - Tests generated independently from code, when the specifications are fresh in the mind of analysts
  - The generation of test cases may highlight inconsistencies and incompleteness of the corresponding specifications
  - tests may be used as compendium of the specifications by the programmers

(c) 2007 Mauro Pezzè & Michal Young

Improving the Process

- Long lasting errors are common
- It is important to structure the process for
  - Identifying the most critical persistent faults
  - tracking them to frequent errors
  - adjusting the development and quality processes to eliminate errors
- Feedback mechanisms are the main ingredient of the quality process for identifying and removing errors

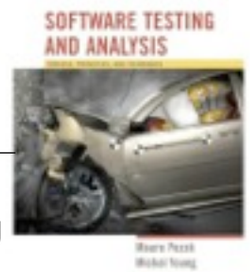
(c) 2007 Mauro Pezzè & Michal Young



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

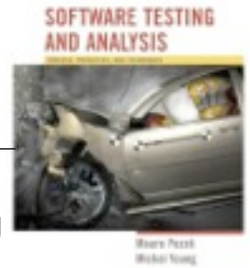
(c) 2007 Mauro Pezzè & Michal Young



Example of Allocation of Responsibilities

- Allocating tasks and responsibilities is a complex job:
  - 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

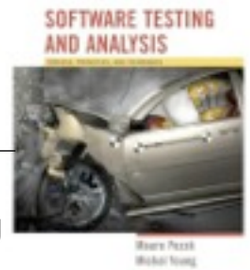
(c) 2007 Mauro Pezzè & Michal Young



Allocation of Responsibilities and rewarding mechanisms: case A

- allocation of responsibilities
  - Development team responsible development measured 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

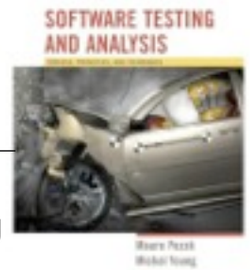
(c) 2007 Mauro Pezzè & Michal Young



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

(c) 2007 Mauro Pezzè & Michal Young



Summary

- Test and Analysis are complex activities that must be suitably planned and monitored
- A good quality process obeys some basic principles:
  - visibility
  - early activities
  - feedback
- aims at
  - reducing occurrences of faults
  - assessing the product dependability before delivery
  - improving the process

(c) 2007 Mauro Pezzè & Michal Young