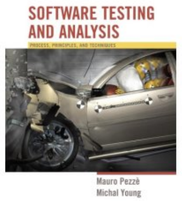
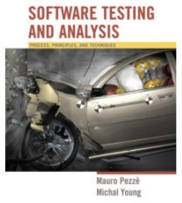
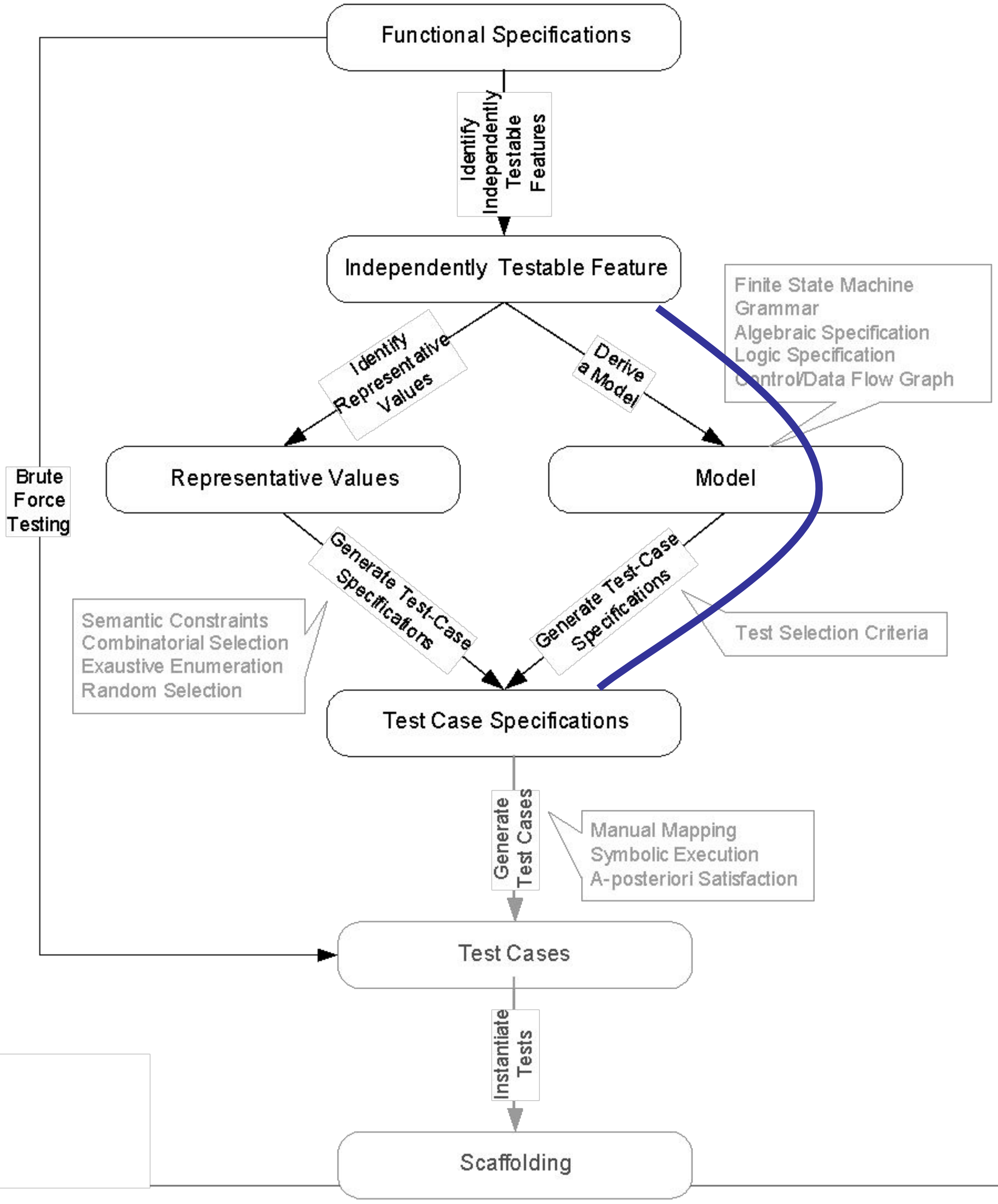
Model based testing

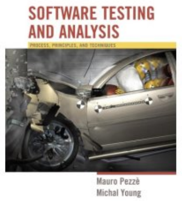


Learning Objectives

- Understand the role of models in devising test cases
    - Principles underlying functional and structural test adequacy criteria, as well as model-based testing
- Understand some examples of model-based testing techniques
    - A few of the most common model-based techniques, representative of many others
- Be able to understand, devise and refine other model-based testing techniques
    - Grasp the basic approach and rationale well enough to apply it in other contexts

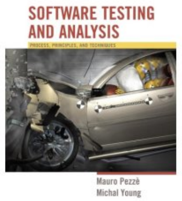



Functional Specifications
Identify Independently Testable Features
Independently Testable Feature
Identify Representative Values
Derive a Model
Finite State Machine
Grammar
Algebraic Specification
Logic Specification
Control/Data Flow Graph
Brute Force Testing
Representative Values
Model
Generate Test-Case Specifications
Generate Test-Case Specifications
Semantic Constraints
Combinatorial Selection
Exaustive Enumeration
Random Selection
Test Selection Criteria
Test Case Specifications
Generate Test Cases
Manual Mapping
Symbolic Execution
A-posteriori Satisfaction
Test Cases
Instantiate Tests
Scaffolding



Why model-based testing?

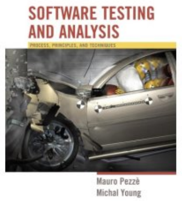
- Models used in specification or design have structure
    - Useful information for selecting representative classes of behavior; behaviors that are treated differently with respect to the model should be tried by a thorough test suite
    - In combinatorial testing, it is difficult to capture that structure clearly and correctly in constraints
- We can devise test cases to check actual behavior against behavior specified by the model
    - “Coverage” similar to structural testing, but applied to specification and design models



Deriving test cases from finite state machines

A common kind of model for describing behavior that depends on sequences of events or stimuli

Example: UML state diagrams



From an informal specification…

Maintenance: The Maintenance function records the history of items undergoing maintenance.

If the product is covered by warranty or mainte
requested either by calling the maintenance to or
by bringing the item to a designated maintenance station

If the maintenance is requested by phone or we
resident, the item is picked up at the customer site, otherwise, the customer shall ship the item with an express courier.

If the maintenance contract number provided b ows
the procedure for items not covered by warran

If the product is not covered by warranty or mainte
requested only by bringing the item to a mainte
informs the customer of the estimated costs for repair. Maintenance starts only when the customer accepts the estimate.

If the customer does not accept the estimate, the produc

Small problems can be repaired directly at the
station cannot solve the problem, the product
headquarters (if in US or EU) or to the maintenance main headquarters (otherwise).

If the maintenance regional headquarters cannot solve the problem, the product is sent to the maintenance main headquarters.

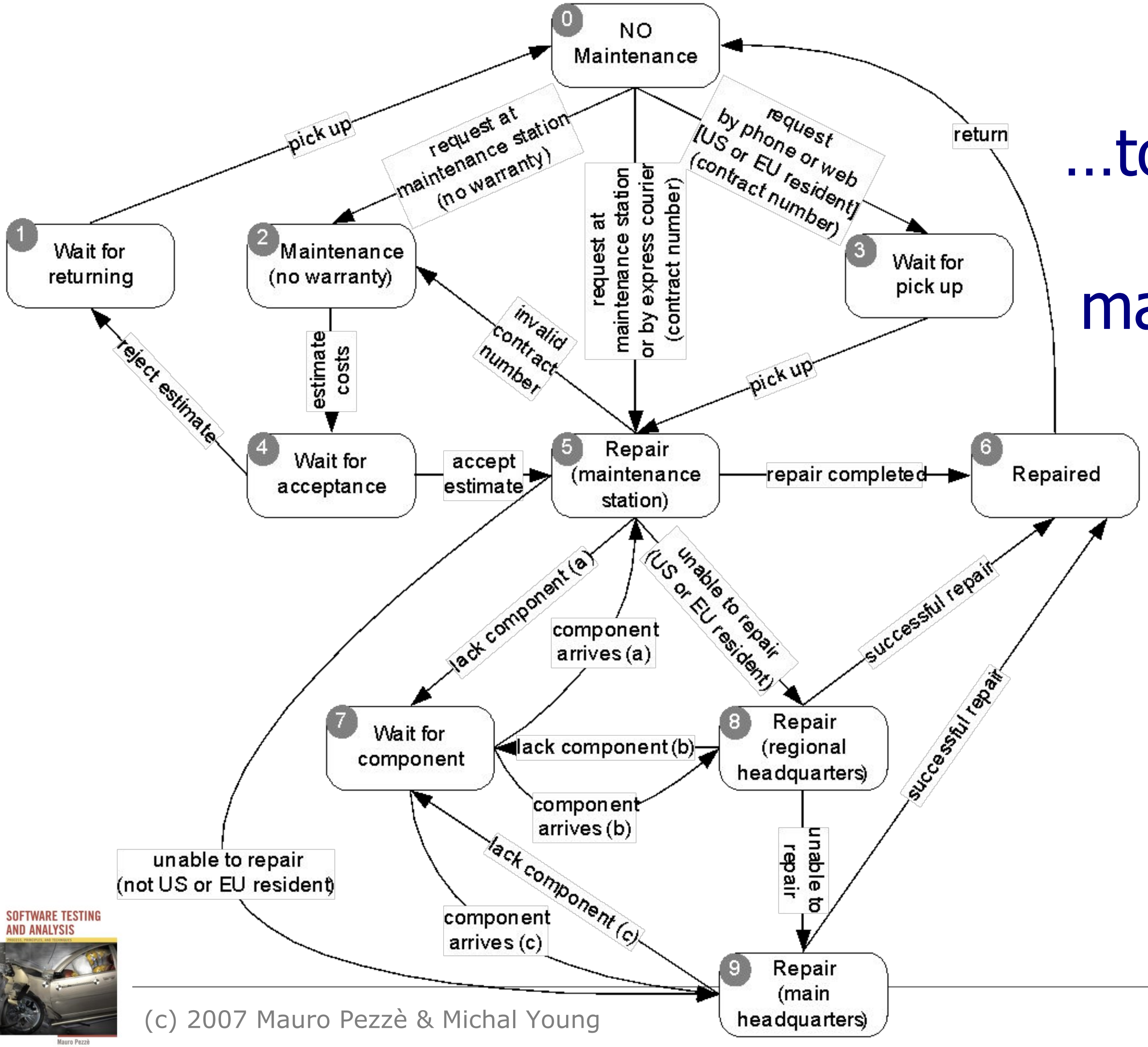
Maintenance is suspended if some components are not available.

Once repaired, the product is returned to the customer.

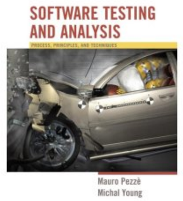
Multiple choices in the first step ...

... determine the possibilities for the next step ...

... and so on ...



…to a finite state machine…

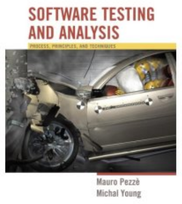


…to a test suite

Meaning: From state 0 to state 2 to state 4 to state 1 to state 0

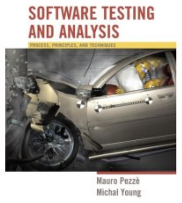
TC1	0	2	4	1	0						
TC2	0	5	2	4	5	6	0				
TC3	0	3	5	9	6	0					
TC4	0	3	5	7	5	8	7	8	9	6	0

Is this a thorough test suite?
How can we judge?



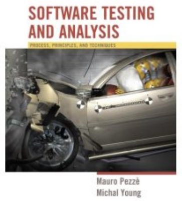
“Covering” finite state machines

- State coverage:
  - Every state in the model should be visited by at least one test case
- Transition coverage
  - Every transition between states should be traversed by at least one test case.
  - *This is the most commonly used criterion*
    - A transition can be thought of as a (precondition, postcondition) pair



Path sensitive criteria?

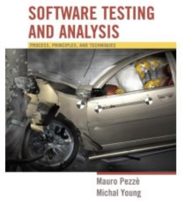
- Basic assumption: States fully summarize history
    - No distinction based on how we reached a state; this should be true of well-designed state machine models
- If the assumption is violated, we may distinguish paths and devise criteria to cover them
    - Single state path coverage:
        - traverse each subpath that reaches each state at most once
    - Single transition path coverage:
        - traverse each transition at most once
    - Boundary interior loop coverage:
        - each distinct loop of the state machine must be exercised the minimum, an intermediate, and the maximum or a large number of times
        - *Of the path sensitive criteria, only boundary-interior is common*



Testing decision structures

Some specifications are structured as decision tables, decision trees or flow charts.

We can exercise these as if they were program source code.



From an informal specification..

Pricing: The pricing function determines the adjusted price of a configuration for a particular customer.

The scheduled price of a configuration is the sum of the scheduled price of the model and the scheduled price of each component in the configuration. The adjusted price is either the scheduled price, if no discounts are applicable, or the scheduled price less any applicable discounts.

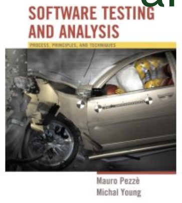
There are three price schedules and three corresponding discount schedules, Business, Educational, and Individual.

….

- Educational prices: The adjusted price for a purchase charged to an educational account in good standing is the scheduled price from the educational price schedule. No further discounts apply.

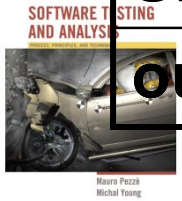
…

- Special-price non-discountable offers: Sometimes a complete configuration is offered at a special, non-discountable price. When a special, non-discountable price is available for a configuration, the adjusted price is the non-discountable price or the regular price after any applicable discounts, whichever is less



…to a decision table …

	edu		individual					
EduAc	T	T	F	F	F	F	F	F
BusAc	-	-	F	F	F	F	F	F
CP > CT1	-	-	F	F	T	T	-	-
YP > YT1	-	-	-	-	-	-	-	-
CP > CT2	-	-	-	-	F	F	T	T
YP > YT2	-	-	-	-	-	-	-	-
SP < Sc	F	T	F	T	-	-	-	-
SP < T1	-	-	-	-	F	T	-	-
SP < T2	-	-	-	-	-	-	F	T
out	Edu	SP	ND	SP	T1	SP	T2	SP



…with constraints…

at-most-one (EduAc, BusAc)

at-most-one (YP < YT1, YP > YT2)

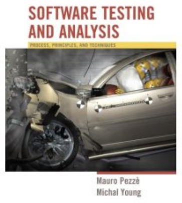
YP > YT2 -> YP > YT1

at-most-one (CP < CT1, CP > CT2)

CP > CT2 -> CP > CT1

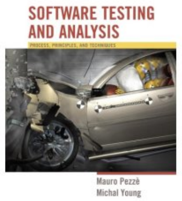
at-most-one (SP < T1, SP > T2

SP > T2 -> SP > T1



…to test cases

- Basic condition coverage
  - a test case specification for each column in the table
- Compound condition adequacy criterion
  - a test case specification for each combination of truth values of basic conditions
- Modified condition/decision adequacy criterion (MC/DC)
  - each column in the table represents a test case specification.
  - we add columns that differ in one input row and in outcome, then merge compatible columns



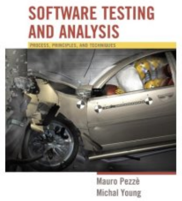
Example MC/DC

	C.1	C.1a	C.1b	C.1
EduAc	T	F	T	-
BusAc	-	-	-	T
CP > CT1	-	-	-	F
YP > YT1	-	-	-	F
CP > CT2	-	-	-	-
YP > YT2	-	-	-	-
SP > Sc	F	F	T	T
SP > T1	-	-	-	-
SP > T2	-	-	-	-
out	Edu	*	*	SP

Generate C.1a and C.1b by flipping one element of C.1

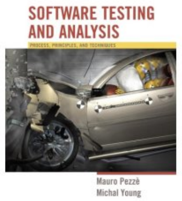
C.1b can be merged with an existing column (C.10) in the spec

Outcome of generated columns must differ from source column



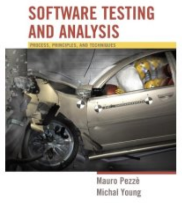
Flowgraph based testing

If the specification or model has both decisions and sequential logic, we can cover it like program source code.



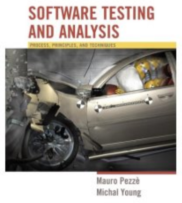
from an informal spec (i/iii)...

- Process shipping order: The Process shipping order function checks the validity of orders and prepares the receipt A valid order contains the following data:
  - cost of goods: If the cost of goods is less than the minimum processable order (MinOrder) then the order is invalid.
  - shipping address: The address includes name, address, city, postal code, and country.
  - preferred shipping method: If the address is domestic, the shipping method must be either land freight, expedited land freight, or overnight air; If the address is international, the shipping method must be either air freight, or expedited air freight.



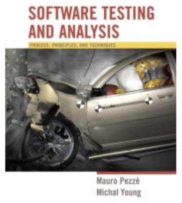
…(ii/iii)…

- a shipping cost is computed based on
  - address and shipping method.
  - type of customer which can be individual, business, educational
- preferred method of payment. Individual customers can use only credit cards, business and educational customers can choose between credit card and invoice
- card information: if the method of payment is credit card, fields credit card number, name on card, expiration date, and billing address, if different than shipping address, must be provided. If credit card information is not valid the user can either provide new data or abort the order

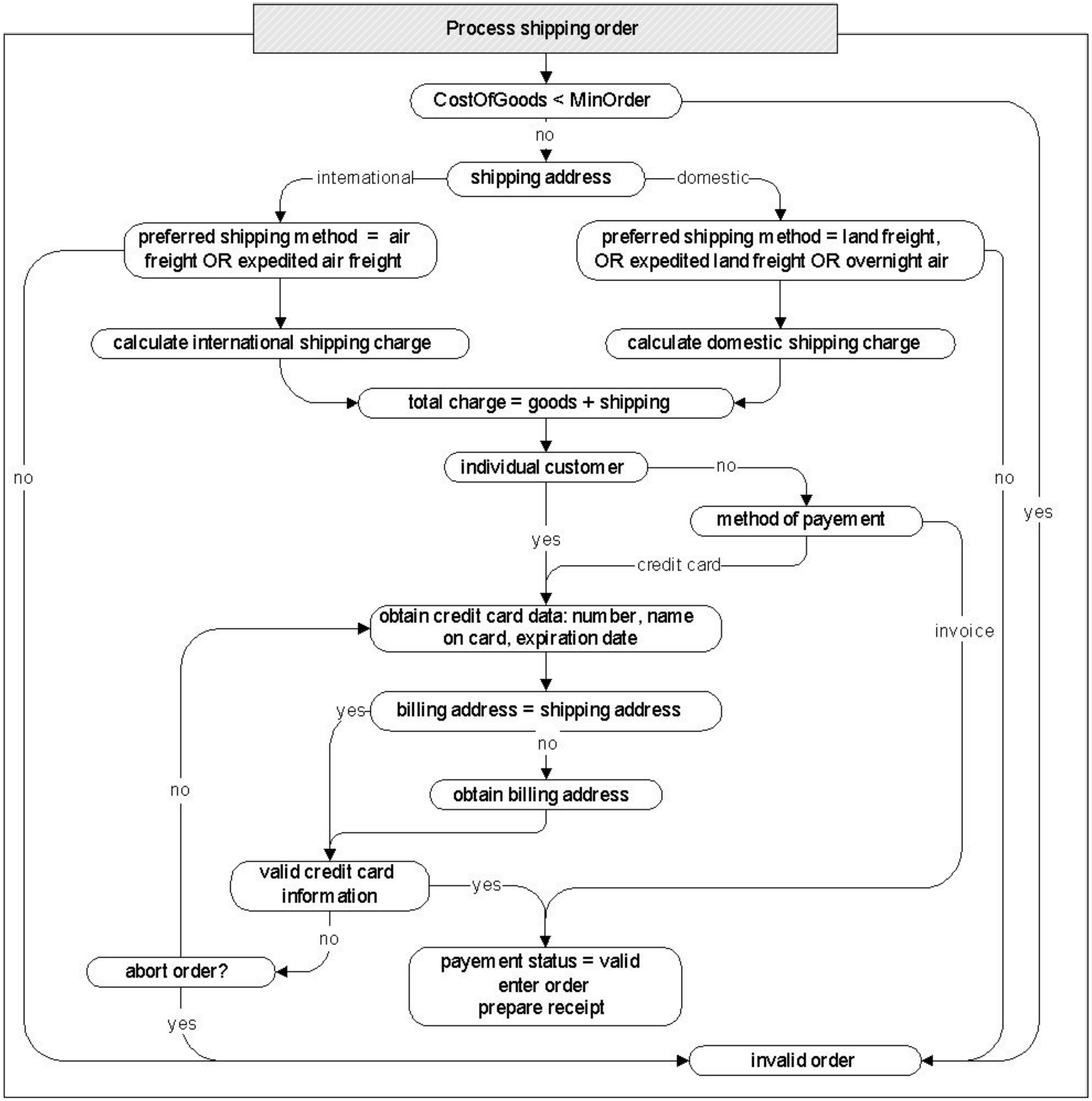


…(iii/iii)

- The outputs of Process shipping order are
- validity: Validity is a boolean output which indicates whether the order can be processed.
- total charge: The total charge is the sum of the value of goods and the computed shipping costs (only if validity = true).
- payment status: if all data are processed correctly and the credit card information is valid or the payment is invoice, payment status is set to valid, the order is entered and a receipt is prepared; otherwise validity = false.



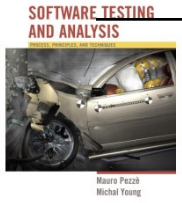
…to a flowgraph


Process shipping order
CostOfGoods < MinOrder
no
yes
shipping address
international
domestic
preferred shipping method = air freight OR expedited air freight
preferred shipping method = land freight, OR expedited land freight OR overnight air
no
no
calculate international shipping charge
calculate domestic shipping charge
total charge = goods + shipping
individual customer
no
yes
method of payement
credit card
invoice
obtain credit card data: number, name on card, expiration date
billing address = shipping address
yes
no
obtain billing address
valid credit card information
yes
no
abort order?
no
yes
payement status = valid enter order prepare receipt
invalid order

...from the flow graph to test cases

Branch testing: cover all branches

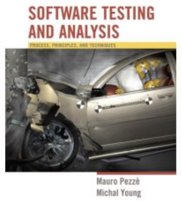
Case	Too Small	Ship Where	Ship Method	Cust Type	Pay Method	Same Address	CC valid
TC-1	No	Int	Air	Bus	CC	No	Yes
TC-2	No	Dom	Land	-	-	-	-
TC-3	Yes	-	-	-	-	-	-
TC-4	No	Dom	Air	-	-	-	-
TC-5	No	Int	Land	-	-	-	-
TC-6	No	-	-	Edu	Inv	-	-
TC-7	No	-	-	-	CC	Yes	-
TC-8	No	-	-	-	CC	-	No (abort)
TC-9	No	-	-	-	CC	-	No (no abort)



Grammar-based testing

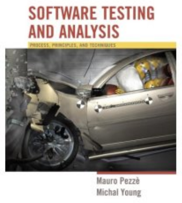
Complex input is (or can) often be described

by a context-free grammar



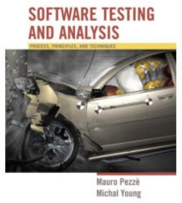
Grammars in specifications

- Grammars are good at:
  - Representing inputs of varying and unbounded size
  - With recursive structure
  - And boundary conditions
- Examples:
  - Complex textual inputs
  - Trees (search trees, parse trees, ...)
    - Note XML and HTMl are trees in textual form
  - Program structures
    - Which are also tree structures in textual format!



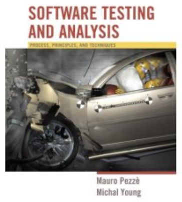
Grammar-based testing

- Test cases are strings *generated* from the grammar
- Coverage criteria:
  - Production coverage: each production must be used to generate at least one (section of) test case
  - Boundary condition: annotate each recursive production with minimum and maximum number of application, then generate:
    - Minimum
    - Minimum + 1
    - Maximum - 1
    - Maximum



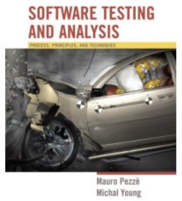
from an informal specification (i/iii)…

- The Check-configuration function checks the validity of a computer configuration.
- The parameters of check-configuration are:
  - Model
  - Set of components


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

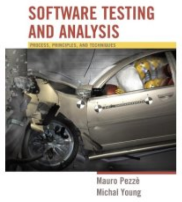
… (ii/iii)…

- Model: A model identifies a specific product and determines a set of constraints on available components. Models are characterized by logical slots for components, which may or may not be implemented by physical slots on a bus. Slots may be required or optional. Required slots must be assigned with a suitable component to obtain a legal configuration, while optional slots may be left empty or filled depending on the customers' needs
  - Example: The required ``slots'' of the Chipmunk C20 laptop computer include a screen, a processor, a hard disk, memory, and an operating system. (Of these, only the hard disk and memory are implemented using actual hardware slots on a bus.) The optional slots include external storage devices such as a CD/DVD writer.



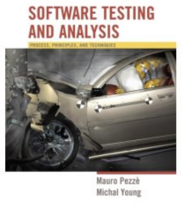
… (iii/iii)

- Set of Components: A set of [slot, component] pairs, which must correspond to the required and optional slots associated with the model. A component is a choice that can be varied within a model, and which is not designed to be replaced by the end user. Available components and a default for each slot is determined by the model. The special value empty is allowed (and may be the default selection) for optional slots. In addition to being compatible or incompatible with a particular model and slot, individual components may be compatible or incompatible with each other.



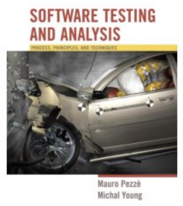
…to a grammar

```
<Model>                 ::= <modelNumber> <compSequence>
                            <optCompSequence>
<compSequence>          ::= <Component> <compSequence> | empty
<optCompSequence>       ::= <OptionalComponent> <optCompSequence> |
                            empty
<Component>             ::= <ComponentType> <ComponentValue>
<OptionalComponent>     ::= <ComponentType>
<modelNumber>           ::= string
<ComponentType>         ::= string
<ComponentValue>        ::= string
```



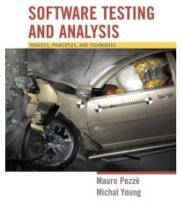
…to a grammar with limits

Model	<Model>	::= <modelNumber> <compSequence> <optCompSequence>
compSeq1 [0, 16]	<compSequence>	::= <Component> <compSequence>
compSeq2	<compSequence>	::= empty
optCompSeq1 [0, 16]	<optCompSequence>	::= <OptionalComponent> <optCompSequence>
optCompSeq2	<optCompSequence>	::= empty
Comp	<Component>	::= <ComponentType> <ComponentValue>
OptComp	<OptionalComponent>	::= <ComponentType>
modNum	<modelNumber>	::= string
CompTyp	<ComponentType>	::= string
CompVal	<ComponentValue>	::= string



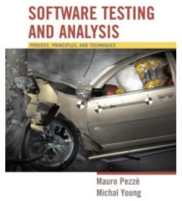
…to test cases

- "Mod000"
  - Covers Model, compSeq1[0], compSeq2, optCompSeq1[0], optCompSeq2, modNum
- "Mod000 (Comp000, Val000) (OptComp000)"
  - Covers Model, compSeq1[1], compSeq2, optCompSeq2[0], optCompSeq2, Comp, OptComp, modNum, CompTyp, CompVal
- Etc…
- Comments:
  - By first applying productions with nonterminals on the right side, we obtain few, large test cases
  - By first applying productions with terminals on the right side, we obtain many, small test cases



Grammar vs. Combinatorial Testing

- Combinatorial specification-based testing is good for "mostly indepedendent" parameters
  - We can incorporate a few constraints, but complex constraints are hard to represent and use
  - We must often "factor and flatten"
    - E.g., separate "set of slots" into characteristics "number of slots" and predicates about what is in the slots (all together)
- Grammar describes sequences and nested structure naturally
  - But some relations among different parts may be difficult to describe and exercise systematically, e.g., compatibility of components with slots



Summary: The big picture

- Models are useful abstractions
  - In specification and design, they help us think and communicate about complex artifacts by emphasizing key features and suppressing details
  - Models convey structure and help us focus on one thing at a time
- We can use them in systematic testing
  - If a model divides behavior into classes, we probably want to exercise each of those classes!
  - Common model-based testing techniques are based on state machines, decision structures, and grammars
    - but we can apply the same approach to other models