Introduction to Software Testing
Chapter 8.4
Logic Coverage for Specifications

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Specifications in Software

- Specifications can be formal or informal
  - Formal specs are usually expressed mathematically
  - Informal specs are usually expressed in *natural language*
- Lots of formal languages and informal styles are available
- Most specification languages include explicit logical expressions, so it is very easy to apply logic coverage criteria
- Implicit logical expressions in natural-language specifications should be re-written as explicit logical expressions as part of test design
  - You will often find mistakes
- One of the most common is preconditions …

Preconditions

- Programmers often include preconditions for their methods
- The preconditions are often expressed in comments in method headers
- Preconditions can be in javadoc, “requires”, “pre”, …

Example – Saving addresses

// name must not be empty
// state must be valid
// zip must be 5 numeric digits
// street must not be empty
// city must not be empty

Conjunctive Normal Form

Rewriting to logical expression

name != “” ∧ state in stateList ∧ zip >= 00000 ∧ zip <= 99999 ∧ street != “” ∧ city != “”

Shortcut for Predicates in Conjunctive Normal Form

- A predicate is in conjunctive normal form (CNF) if it consists of clauses or conjuncts connected by the and operator
  - $A \wedge B \wedge C \wedge \ldots$
  - $(A \vee B) \wedge (C \vee D)$
- A major clause is made active by making all other clauses true
- ACC tests are "all true" and then a "diagonal" of false values:

	A	B	C	…
1	T	T	T	
2	F	T	T	…
3	T	F	T	
4	T	T	F	
		.		.
		.		.
		.		.

Shortcut for Predicates in Disjunctive Normal Form

- A predicate is in disjunctive normal form (DNF) if it consists of clauses or conjuncts connected by the or operator
  - $A \lor B \lor C \lor \ldots$
  - $(A \land B) \lor (C \land D)$
- A major clause is made active by making all other clauses false
- ACC tests are "all false" and then a "diagonal" of true values:

	A	B	C	…
1	F	F	F	
2	T	F	F	…
3	F	T	F	
4	F	F	T	
		.		.
		.		.
		.		.

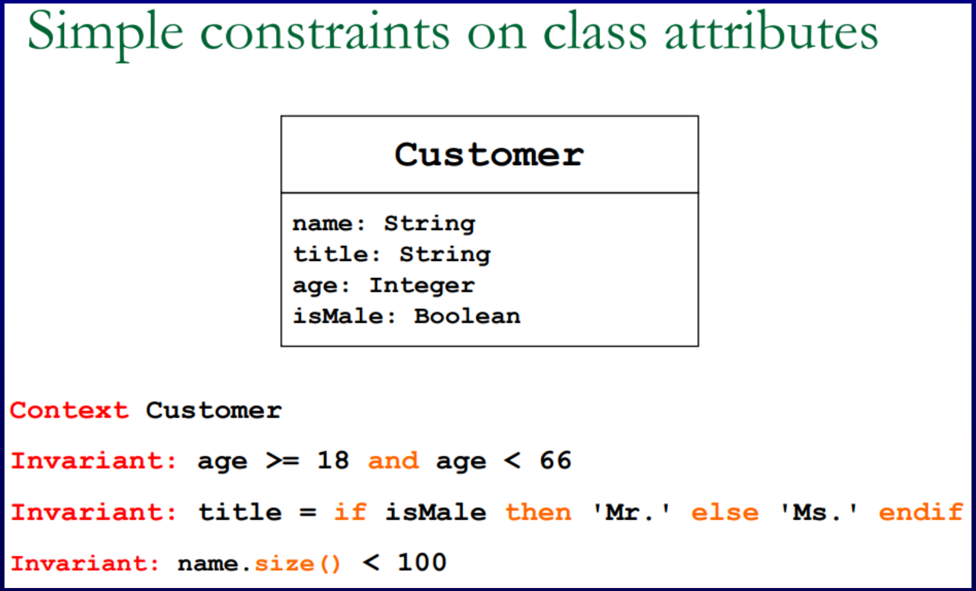
Summary : Logic Coverage for Specs

- Logical specifications can come from lots of places :
  - Preconditions
  - Java asserts
  - Contracts (in design-by-contract development)
  - OCL conditions
  - Formal languages
- Logic specifications can describe behavior at many levels :
  - Methods and classes (unit and module testing)
  - Connections among classes and components
  - System-level behavior
- Many predicates in specifications are in disjunctive normal or conjunctive normal form—simplifying the computations

Further Reading

OCL (Object Constraint Language)

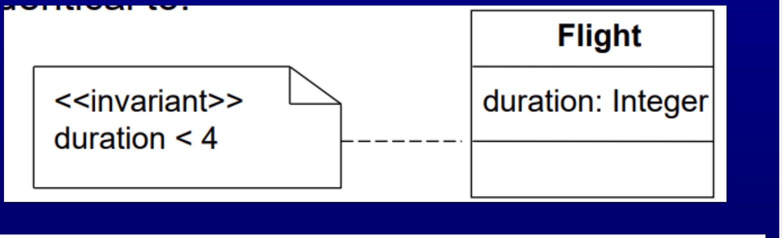
- Object Constraint Language (OCL) is a formal language used to describe expressions on UML models
- These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model



Simple constraints on class attributes

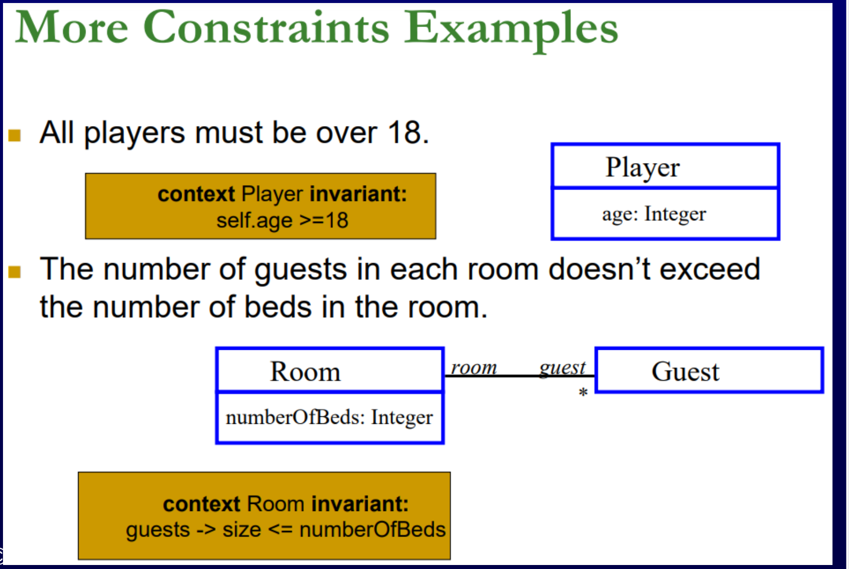
```
Customer
name: String
title: String
age: Integer
isMale: Boolean
```

```
Context Customer
Invariant: age >= 18 and age < 66
Invariant: title = if isMale then 'Mr.' else 'Ms.' endif
Invariant: name.size() < 100
```



<<invariant>>
duration < 4

Flight
duration: Integer



More Constraints Examples

- All players must be over 18.

Player
age: Integer

context Player **invariant:**
self.age >=18

- The number of guests in each room doesn't exceed the number of beds in the room.

Room
numberOfBeds: Integer

Guest

context Room **invariant:**
guests -> size <= numberOfBeds

- https://www.omg.org/spec/OCL/2.4/PDF

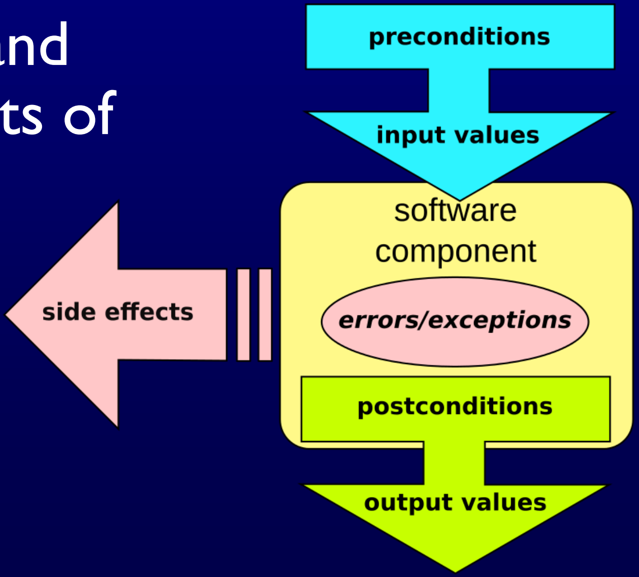
Design by contract (DbC)

- Design by contract (DbC)
  - Also known as:
    - contract programming
    - programming by contract
    - design-by-contract programming
  - It is an approach for designing software
  - It prescribes that software designers should define formal, precise and verifiable interface specifications for software components
    - Establishment of well-defined interface parameters, or contracts, for all parts of a program
  - It can be viewed as a conceptual development style that is implemented through documentation or modeling

Meyer, Bertrand: Applying "Design by Contract", in Computer (IEEE), 25, 10, October 1992, pp. 40–51

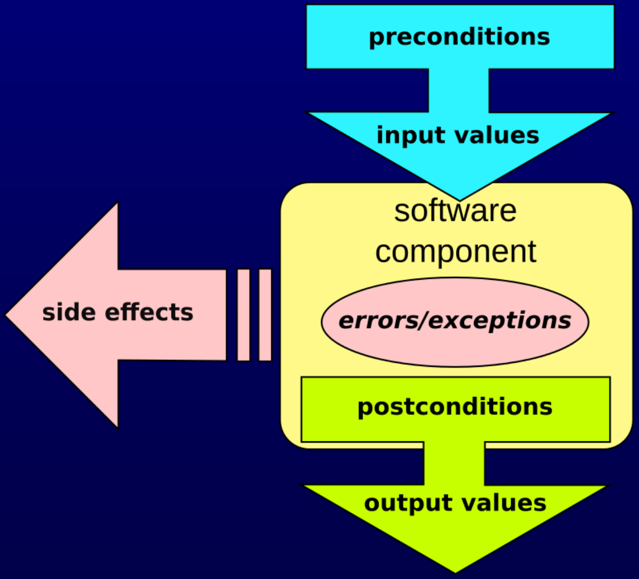
Design by contract (DbC)

- Design by contract (DbC)
  - In this model, each method and class in an object oriented program defines a contract by which any other method or object interacting with it must abide
  - These two parts of the DbC model define the state of the program before a method is called and the state of the program after the method has completed executing
  - By developing contracts for each class and method, the interaction of different parts of a program can easily be predicted



Design by contract (DbC)

- Design by contract (DbC)
  - It ensures that the program will not attempt to execute if there is a violation of contracts
    - Because any output produced in that state would technically be invalid anyway
  - Note that when the expressions are evaluated, they can have side effects (i.e., their evaluation can modifies some state variable value(s) outside its local environment, which is to say if it has any observable effect other than its primary effect of returning a value to the caller)


preconditions
input values
software component
side effects
errors/exceptions
postconditions
output values