# Specifying Operations

Based on Chapter 10 of Bennett, McRobb and Farmer:

*Object Oriented Systems Analysis and Design Using UML,* (4th Edition), McGraw Hill, 2010.

# In This Lecture You Will Learn:

- Why operations need to be specified
- What is meant by “Contracts”
- Non-algorithmic ways of describing operations:
  - Decision Tables
  - Pre- and Post-Condition Pairs
- Algorithmic ways of describing operations:
  - Structured English and Pseudocode
  - Activity Diagrams
  - Object Constraint Language

# Why We Specify Operations

- From analysis perspective:
  - Ensure users' needs are understood
- From design perspective:
  - Guide programmer to an appropriate implementation (i.e. method)
- From test perspective:
  - Verify that the method does what was originally intended

# Operations and Their Effects

- Operations with side-effects may:
  - Create or destroy object instances
  - Set attribute values
  - Form or break links with other objects
  - Carry out calculations
  - Send messages or events to other objects
  - Any combination of these
- Some operations have no side-effects:
  - They return data but do not change anything

# Services Among Objects

- When objects collaborate, one object typically provides a service to another
- Examples:
  - A `Client` object might ask a `Campaign` object for its details
  - The same `Client` object might then ask a boundary object to display its related `Campaign` details to the user

# Contracts: an Approach to Defining Services

- A service can be defined as a contract between the participating objects
- Contracts focus on inputs and outputs
- The intervening process is seen as a black box, with irrelevant details hidden
- This emphasises service delivery, and ignores implementation

# Contract-Style Operation Specification

- Intent / purpose of the operation
- Operation signature, including return type
- Description of the logic
- Other operations called
- Events transmitted to other objects
- Any attributes set
- Response to exceptions (e.g. an invalid parameter)
- Non-functional requirements

(adapted from Larman, 2005 and Allen and Frost, 1998)

# Types of Logic Specification

- Logic description is probably the most important element
- Two main categories:
- ***Non-algorithmic*** methods focus on ***what*** the operation should achieve—black box approach
- ***Algorithmic*** types focus on ***how*** the operation should work —white box approach

# Non-Algorithmic Techniques

- Use when correct result matters more than the method used to reach it…
- …Or when no decision has yet been made about the best method
  - Decision tree: complex decisions, multiple criteria and steps (not described further here)
  - Decision table: similar applications to decision tree
  - Pre- and Post-Condition Pairs: suitable where precise logic is unimportant / uncertain

# Decision Table

- Many variants, but all work by identifying:
  - Combinations of initial conditions = ‘rules’
  - Outcomes that should result depending on what conditions are true = ‘actions’
- Rules and actions are displayed in tabular form

# Example Decision Tree

**Conditions to be tested**

| Conditions and actions | Rule 1 | Rule 2 | Rule 3 |
|---|---|---|---|
| **Conditions** | | | |
| Is budget likely to be overspent? | N | Y | Y |
| Is overspend likely to exceed 2%? | - | N | Y |
| **Actions** | | | |
| No action | X | | |
| Send letter | | X | X |
| Set up meeting | | | X |

**Possible actions**

# Pre- / Post-Condition Pair

- Logically similar to decision table
- Identifies conditions that:
  - …must be true for operation to execute = pre-conditions
  - …must be true *after* operation has executed = post-conditions
- May be written in formal language (e.g. OCL)

# Pre- / Post-Condition Pair:
# `Change staff grade`

*pre-conditions:*

- `creativeStaffObject` is valid
- `gradeObject` is valid
- `gradeChangeDate` is a valid date
- `gradeChangeDate` is greater than or equal to today's date

*post-conditions:*

- a new `staffGradeObject` exists
- new `staffGradeObject` linked to `creativeStaffObject`
- new `staffGradeObject` linked to previous
- value of previous `staffGradeObject.gradeFinishDate` set equal to `gradeChangeDate - 1 day`

# Algorithmic Techniques

- Suitable where a decision can be made about the best method to use
- Can be constructed top-down, to handle arbitrarily complex functionality
- Examples:
  - Structured English
  - Activity Diagrams

# Structured English

- Commonly used, easy to learn
- Three types of control structure, derived from structured programming:
  - Sequences of instructions
  - Selection of alternative instructions (or groups of instruction)
  - Iteration (repetition) of instructions (or groups)

# Sequence in Structured English

- Each instruction is executed in turn, one after another:

```
get client contact name
sale cost = item cost * ( 1 - discount rate )
calculate total bonus
description = new description
```

# Selection in Structured English

- One or other alternative course is followed, depending on result of a test:

```
if client contact is 'Sushila'
  set discount rate to 5%
else
  set discount rate to 2%
end if
```

# Iteration in Structured English

- Instruction or block of instructions is repeated
  - Can be a set number of repeats
  - Or until some test is satisfied:

```
do while there are more staff in the list
  calculate staff bonus
  store bonus amount
end do
```

# Structured English can be Arbitrarily Complex

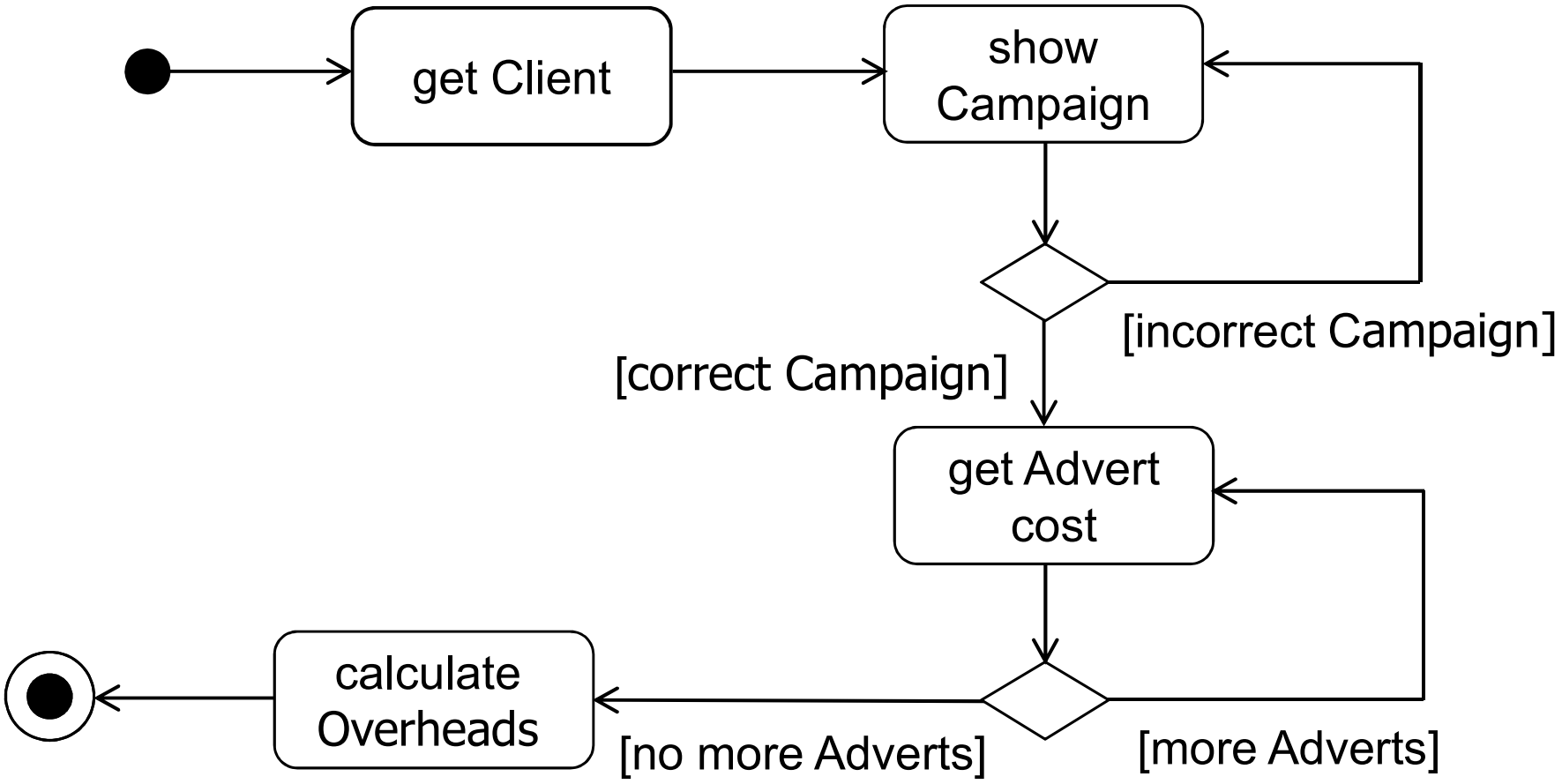
```
do while there are more staff in the list
  calculate bonus for this staff member
  begin case
     case bonus > £250
          add name to 'star of month' list
     case bonus < £25
          create warning letter
  end case
  store bonus amount
end do
format bonus list
```

```
do while there are more adverts for campaign
  get next advert
  get cost for this advert
  add to cumulative cost for campaign
end do
set total advert cost = final cumulative
  cost
set total campaign cost =  total advert cost
  + (total advert cost X overhead rate)
get campaign budget
if total campaign cost > campaign budget
  generate warning
endif
```

# Activity Diagrams

- Part of UML notation set
- Can be used for operation logic specification, among many other uses
- Easy to learn and understand
- Has the immediacy of graphic notation
- Some resemblance to old-fashioned flowchart technique

# Example Activity Diagram:
## Check campaign budget



# Object Constraint Language

- A formal language used for:
  - Precise definition of constraints on model elements
  - E.g. pre- and post-conditions of operations
- OCL statements can:
  - Define queries
  - Reference values
  - State business rules

# Object Constraint Language

- Most OCL statements consist of:
- Context, Property and Operation
- *Context*
  - Defines domain within which expression is valid
  - Instance of a type, e.g. object in class diagram
  - Link (association instance) may be a context
- A *property* of that instance
  - Often an attribute, association-end or query operation

- OCL *operation* is applied to the property
- Operations include
  - Arithmetical operators `*`, `+`, `-` and `/`
  - Set operators such as `size`, `isEmpty` and `select`
  - Type operators such as `oclIsTypeOf`

| OCL expression | Interpretation |
|---|---|
| **context** Person<br>self.gender | In the context of a specific person, the value of the property 'gender' of that person—i.e. a person's gender. |
| **context** Person<br>**inv:** self.savings >= 500 | The property 'savings' of the person under consideration must be always be greater than or equal to 500. |
| **context** Person<br>self.husband->notEmpty **implies**<br>self.husband.gender = male | If the set 'husband' associated with a person is not empty, then the value of the property 'gender' of the husband must be male. Boldface denotes OCL keyword, but has no semantic import. |
| **context** Company<br>**inv:** self.CEO->size <= 1 | The size of the set of the property 'CEO' of a company must be less than or equal to 1. That is, a company cannot have more than 1 Chief Executive Officer. |
| **context** Company<br>self.employee->select (age<60) | The set of employees of a company whose age is less than 60. |

# OCL Used for Pre- / Post-Conditions

```
context: CreativeStaff::changeGrade(grade:Grade,
   gradeChangeDate:Date)
pre:
   grade oclIsTypeOf(Grade)
   gradeChangeDate >= today
post:
   self.staffGrade->exists and
   self.staffGrade[previous]->notEmpty and
   self.staffGrade.gradeStartDate = gradeChangeDate and
   self.staffGrade.previous.gradeFinishDate =
   gradeChangeDate - 1 day
```

# Summary

In this lecture you have learned about:

- The role of operation specifications
- What is meant by “Contracts”
- Algorithmic and non-algorithmic techniques, and how they differ
- About the use of:
  - Decision Tables, Pre- and Post-Condition Pairs, Structured English, Activity Diagrams and Object Constraint Language

# References

- Bennett, McRobb and Farmer (2002)
- Yourdon (1989) covers Structured English and Pre- / Post-Conditions well
- Senn (1989) is good on Decision Tables
- Larman (1998) takes a contract-based approach to O-O analysis and design, with examples taken to Java code

  (For full bibliographic details, see Bennett, McRobb and Farmer)