Specifying Operations

Chapter 10

In this Lecture you will Learn:

- About the role of operation specifications
- What is meant by “Contracts”
- About algorithmic and non-algorithmic techniques, and how they differ
- How to use:
  - Decision Tables
  - Pre- and Post-condition Pairs
  - Structured English
  - 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

Main Purposes of Operation Specification

- Confirm that the logic of user requirements has been understood correctly and documented accurately.
- Provide designers and programmers with a detailed and unambiguous basis for the design and implementation of system behavior.

Types of Operation 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
- Operations without side-effects are pure queries
  - Request 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

Class Exercise

- Why are operation specifications in an object-oriented project likely to be small?

Contracts: an Approach to Defining Services

- A service can be defined as a contract between the participating objects
- Contracts focus on inputs and outputs
- Intervening process is seen as a black box
- Irrelevant details are hidden
- This emphasizes service delivery, and ignores implementation

Contract-Style Operation Specification

- Intent or 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

Types of Logic Specification

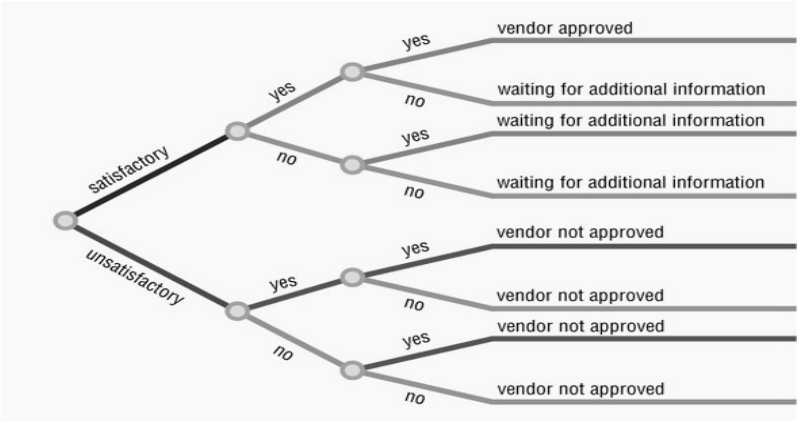
- Logic description is probably the most important element
- Two main categories:
  - **Algorithmic Specification Techniques** describe the sequence of internal logical steps that an operation is to follow.
  - **Non-Algorithmic Techniques** describe the result for a given set of inputs.

Non-Algorithmic Techniques

- Black box type
- Describe the result for a given set of inputs.
- Focus on what the operation should achieve
- Appropriate where correct result matters more than method to arrive at it
  - **Decision Tree**: complex decisions, multiple criteria and steps
  - **Decision Table**: similar applications to decision tree
  - **Pre-condition and Post-condition Pairs**: suitable where precise logic is unimportant or uncertain

Decision Tree

- Diagram that shows conditions and actions graphically



Decision Table

- Decision tables are particularly suited to representing decisions with complex multiple input conditions and complex multiple outcomes, where the precise sequence of steps either is not significant or is not known.

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

Decision Table

- Provides a notation that translates actions and conditions (described in a processing narrative) into a tabular form.
  - The upper left-hand section contains a list of all conditions.
  - The lower left-hand section lists all actions that are possible based on the conditions.
  - The right-hand sections form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination.

Class Exercise

- To what kinds of situation are decision tables particularly suited?

Pre- and Post-condition Pair

- Pre-conditions allow the user and analyst to agree the precise conditions under which it should be possible for an operation to run.
- Post-conditions specify the intended state of the system after the operation has executed.
- This information can be used later by a program designer to specify the operation, by a programmer to code it and by a tester to check whether it works as intended.
- Together, pre- and post-conditions describe an operation's interface to the rest of the system.

Example

- Pre-conditions:
  - `creativeStaffObject` is valid
  - `gradeObject` is valid
  - `gradeChangeDate` is a valid 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`

Class Exercise

- Why is it important to specify both pre- and post-conditions for an operation?

Algorithmic Techniques

- White box type
- Describe the sequence of internal logical steps that an operation is to follow
- Focus on how the operation might work
- Suitable where users understand the procedure for arriving at a result
- Can be constructed top-down, to handle arbitrarily complex functionality
- Examples:
  - Structured English
  - Activity Diagrams

Structured English

- Commonly used, easy to learn
- Also called **Program Design Language** (PDL), or **Pseudocode**.

UPLOADING VENDOR INFORMATION

For each item containing vendor information, perform the following steps:

    If the item is not a computer file then

        Use the scanner to convert it into a file format.

    Copy the file into the Vendor Information folder on your hard disk.

Zip all new files in the Vendor Information folder into a single file.

Save the zipped file in a Web folder.

E-mail the Webmaster with the name of the zipped file.

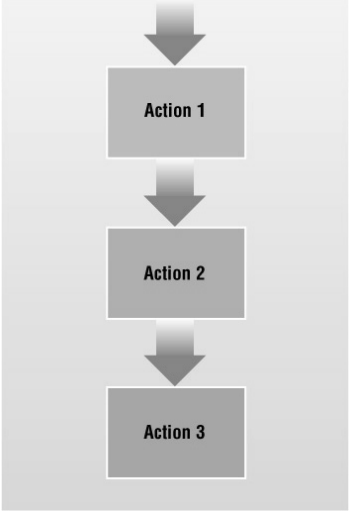
Control Structure in Structured English

- Three types of control structure, derived from structured programming:
  - Sequences of instructions
  - Selection of alternative instructions (or groups of instructions)
  - Iteration (repetition) of instructions (or groups of instructions)

Sequence Structure in Structured English

- Each instruction executed in turn, one after another

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

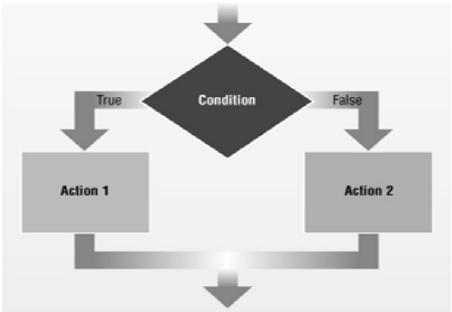


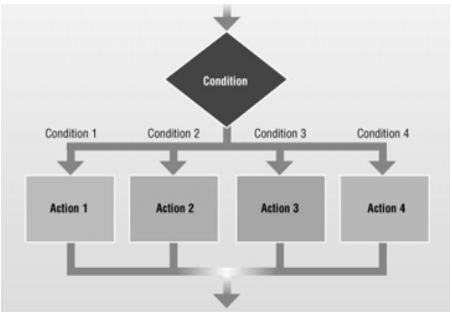
Selection Structure in Structured English

- 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
```


Selection Structure in Structured English



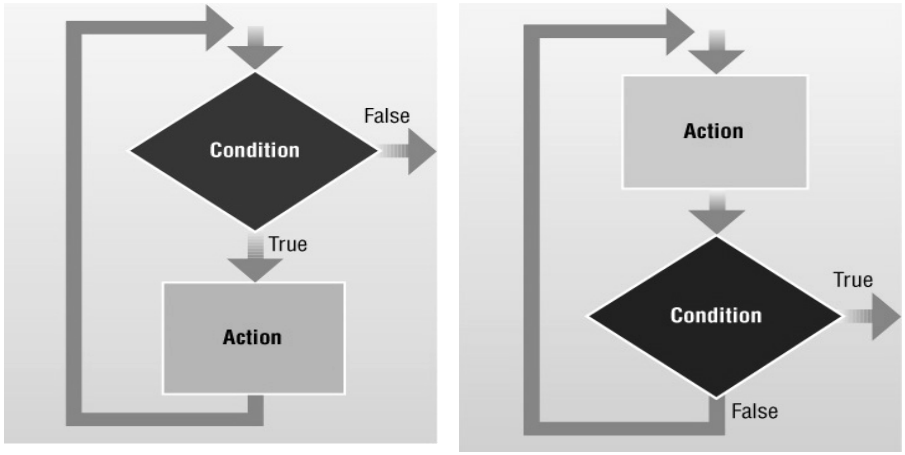


Iteration Structure in Structured English

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

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


Iteration Structure in Structured English

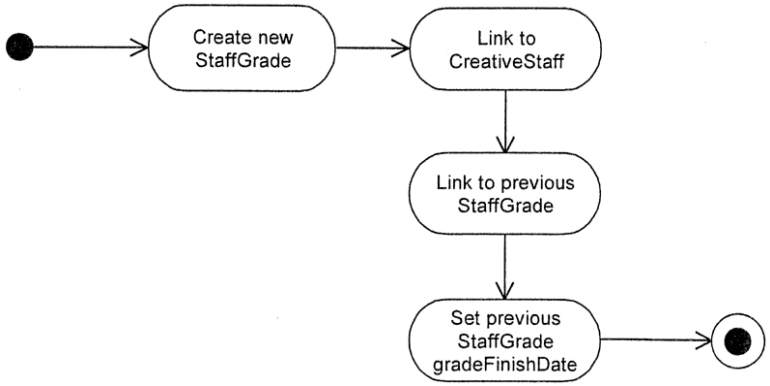


Activity Diagrams

- Activities diagrams can be used to specify the logic of procedurally complex operation.
- Are part of UML notation set
- Can be used for operation logic specification, among many other uses
- Are easy to learn and understand
- Have the immediacy of graphical notation
- Bear some resemblance to old-fashioned flowchart technique

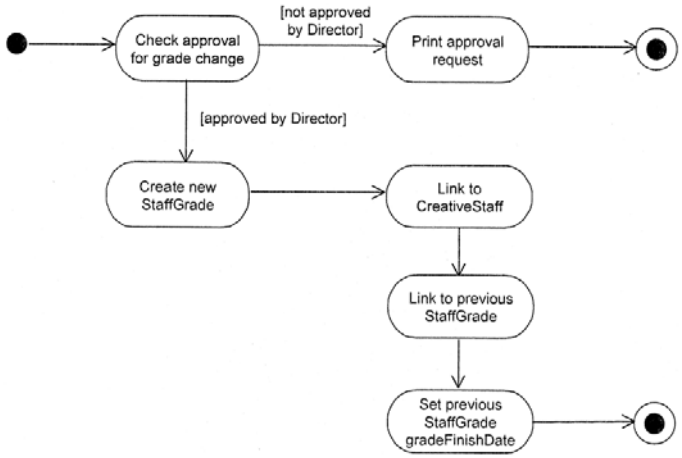
Simple Activity Diagram

- *An activity diagram showing the main steps for the operation* `CreativeStaff.changeGrade()`.



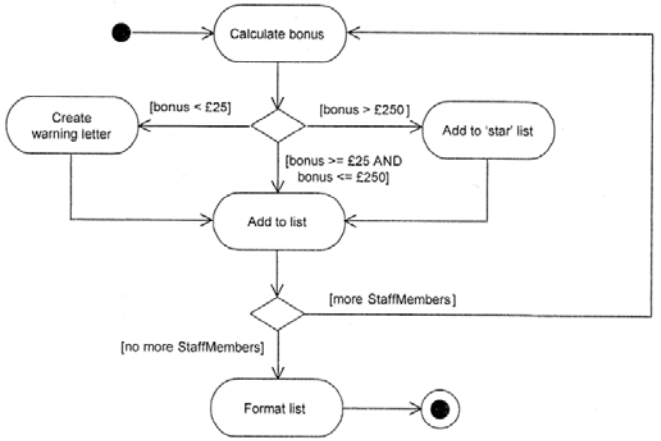
Activity Diagram with Selection

- *A activity diagram for* `CreativeStaff.changeGrade()`*, with an initial selection to check that approval has been given.*



Activity Diagram with Selection and Iteration

- *Activity diagram for* `Prepare bonus list()`*, showing selection and iteration structures.*



Object Constraint Language (OCL)

- OCL expressions usually consist of:
  - A **Context** within which the expression is valid (for example, a specified class);
  - A **Property** within the context to which the expression applies (for example, an attribute of the specified class);
  - An **Operation** that is applied to the property (for example, a mathematical expression that tests the value of the attribute).

Component of OCL - 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

Component of OCL - Property

- A property of that instance
- Often an attribute, association-end or query operation

Component of OCL - 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 and Interpretation

OCL expression	Interpretation
`Person` `  self.gender`	In the context of a specific person, the value of the property 'gender' of that person—i.e. a person's gender.
`Person` `  self.savings >= 500`	The property 'savings' of the person under consideration must be greater than or equal to 500.

`Person`
`  self.husband->notEmpty` **`implies`**
`  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.

`Company` `  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.
`Company` `  self.employee->select (age < 60)`	The set of employees of a company whose age is less than 60.

OCL Used for Pre- and Post-conditions

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

Class Exercise

- Why are non-algorithmic (or declarative) approaches generally preferred in object-oriented development?

Class Exercise

- What is an invariant?