Object Oriented Software
Counting with Multiple Boundaries

Charles Wesolowski  OCUP OCSMP OCRES CFPS
Chief Architect
Special Projects Division
QinetiQ North America Systems Engineering Group
890 Explorer Blvd.
Huntsville, AL 35806
Overview

• Functional Analysis and Architecture
  – Boundaries
• Functional Analysis and Specification
  – UML Analysis Artifacts
• Functional View of an Object
  – Elements of IFPUG Function Point Analysis
• Components
  – Services
Functional Analysis and Architecture

• Functional Analysis organizes information into a model
  – Its “primary purpose is to formulate a model of the problem domain that is independent of implementation considerations.” [UML 2.0 Infrastructure]
  – “The definition of functionality, also referred to as functional analysis, is not the same as structured analysis in software development and does not presume a functionally oriented software design. The definition of functions, their logical groupings, and their association with requirements is referred to as a functional architecture.” [CMMI Second Edition]

• Functional Architecture
  – Specified using a formal language
  – Illustrated using Unified Modeling Language (UML)
  – Measured using ISO Functional Size Method
Calculator Example

- User Requirements -- Four-Function Calculator
- Provide the capability to:
  - Add
  - Subtract
  - Multiply
  - Divide
- Represents an example of stateless software
  - There are no storage requirements
  - All functional size is transactional
Specification and Analysis Artifacts

Boundary Calculate
add : EO
  Inputs
    augend, addend
  Outputs
    sum
subtract : EO
  Inputs
    subtrahend, minuend
  Outputs
    difference
multiply : EO
  Inputs
    multiplicand, multiplier
  Outputs
    product
divide : EO
  Inputs
    dividend, divisor
  Outputs
    quotient
Responses
  DivideByZero

Functional Specification

Interface Models

Multiple Views of a Four-function Calculator
**Action Model of Functions**

A UML Action is “the fundamental unit of executable functionality”

--- All functions have a trigger input

--- Some functions have a response output

Functional Size Methods quantify functionality by accounting for the amount of data a function processes
## Calculator Functional Size

**Boundary: Calculate**

<table>
<thead>
<tr>
<th>Transactional Functions</th>
<th>Type</th>
<th>DET</th>
<th>FTR</th>
<th>Function Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>EO</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>subtract</td>
<td>EO</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>multiply</td>
<td>EO</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>divide</td>
<td>EO</td>
<td>5</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

```java
<<interface>>
Calculate

add(in augend : , in addend : , out sum : )
subtract(in subtrahend : , in minuend : , out difference : )
multiply(in multiplicand : , in multiplier : , out product : )
divide(in dividend : , in divisor : , out quotient : ) : response
```
Elements of Functional Architecture

- **Boundary**
  - A structural element that indicates a partition
    - Represents a group of Functional User Requirements

- **Function**
  - A behavioral element that indicates an operation
    - Represents a Functional User Requirement
    - Characterized by Inputs, Outputs, Responses, Reads, Writes

- **Entity**
  - A structural element that indicates persistent data
    - Represents Domain Objects
    - Organized by groups of data
Functional Size is measured from the User Perspective.

Another software object can play the role of a user.

The type of user has no impact on the functional size of the Object.

At a minimum the functional size of the Object is the size of its elementary processes and the state information that it encapsulates as its internal logical files.
Component View of a Simple Software System

Components are key elements of software architecture that indicate countable software boundaries. They represent the fundamental partitions in Service Oriented Architectures.
Counter Example

- Counter Software – User Requirements
  - The software shall provide an operation to reset the counter to zero.
  - The software shall provide an operation to increment the counter and output its current value.

- Use Case Model
Robustness Analysis Model of Counter

Robustness Analysis links the behavioral model to the persistent data model.
This view assists in identifying FTRs and Data Function types.
Action Model of Counter

Action Models detail data coupling between Elementary Processes And assist in counting logical files
The functional specification contains all information necessary to compute functional size.
All UML analysis artifacts are traced to the specification.
Component models represent reusable functional software entities. The mechanics of constructing and deploying components relates to Software Technical Requirements.
Analysis Reveals Requirements Ambiguity
Component View of Complex Software System

A software system partitioned by non-functional user requirements.

Complex software systems are comprised of coupled components
Each has its own functional size
Components may exhibit behavioral or data dependencies
Behavioral Dependency

An Elementary Process in one object does NOT contribute to the functional size of Another Object.

Elementary Process only count toward the Transactional Function Size of their encapsulating object.
A technical characteristic of object-oriented software requires that an Object's persistent state is accessed via one of its elementary processes, indicated by the <<behavior>> and <<data>> dependencies.

This technical characteristic is irrelevant to the logical view of the software from a functional perspective.
Logical Files in OO Software

An Object provides a Data Function Contribution to Another Object.

In this case, Object appears as Another Logical File to Another User.

The behavior of Object NEVER contributes to the functional size of Another Object.

For that reason, it is not indicated on this diagram.

If Another Elementary Process modifies Another Logical File, it is classified as an ILF, otherwise it is classified as an ELF when computing the functional size of Another Object.
Complex software systems are comprised of coupled components. Each has its own functional size. Components may exhibit behavioral or data dependencies.
Logical File Encapsulated by Object boundary
makes a Data Function Contribution to AnotherObject boundary
Moving Forward

- Components are natural software boundaries
  - Modeled using UML
  - Conform to Object Oriented Software Paradigms
  - Encapsulate Services via Interface Specifications
  - Deployable in Service Oriented Architectures (SOA)

- Components are Countable using the IFPUG method
  - Proven measurement of functional size
  - Applied to industry best practice for software development

- Component Architectures
  - Facilitate communication of requirements to technical staff
  - Provide concrete rather than abstract partitions of functional analysis
Questions????
Backup Slides
Functional analysis of object oriented software requires understanding the techniques used to organize functionality and data within boundaries called objects, using a process of encapsulation that contributes to the cohesion of the software represented by the object.

Encapsulation groups functionality and data in the form of behaviors and persistent state information called an object. Function Point Analysis measures the functionality delivered to the user by describing the object in terms of its Elementary Processes and Logical Files.

An object is defined by a boundary specification called an interface, that contains an enumeration of its functionality expressed as operations.

The user view is dependent on the object’s interface, represented as a boundary class.

Elementary Processes are represented as interface operations.

There is always a behavioral dependency between the user and the software.

A state dependency indicates an instance of a logical file, although objects are not required to maintain state information.

Elementary Processes are realizations of interface operations that may affect the object state, indicated by references to logical files (FTR).

Persistent State information that is maintained by the Object through its Elementary Processes constitutes an Internal Logical File.