CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2009
Doc 15 Cohesion
March 17, 2009

Copyright ©, All rights reserved. 2009 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

Reference

Object Coupling and Object Cohesion, chapter 7 of Essays on Object-Oriented Software Engineering, Vol 1, Berard, Prentice-Hall, 1993,

Cohesion

"Cohesion is the degree to which the tasks performed by a single module are functionally related."

IEEE, 1983

"Cohesion is the "glue" that holds a module together. It can be thought of as the type of association among the component elements of a module. Generally, one wants the highest level of cohesion possible."

Bergland, 1981

"A software component is said to exhibit a high degree of cohesion if the elements in that unit exhibit a high degree of functional relatedness. This means that each element in the program unit should be essential for that unit to achieve its purpose."

Sommerville, 1989

Types of Module Cohesion

Coincidental (worst)

Logical

Temporal

Procedural

Communication

Sequential

Functional (best)

Coincidental Cohesion

Little or no constructive relationship among the elements of the module

Common Object Occurrence

Object does not represent any single object-oriented concept

Collection of commonly used source code as a class inherited via multiple inheritance

```
class Rous
        {
        public static int findPattern( String text, String pattern)
                { // blah}

        public static int average( Vector numbers )
                { // blah}

        public static OutputStream openFile( String fileName )
                { // blah}

        }
```

Logical Cohesion

Module performs a set of related functions, one of which is selected via function parameter when calling the module

Cure – Isolate each function into separate operations

```
public void sample( int flag ){
        switch ( flag ){
                case ON:
                        // bunch of on stuff
                        break;
                case OFF:
                        // bunch of off stuff
                        break;
                case CLOSE:
                        // bunch of close stuff
                        break;
                case COLOR:
                        // bunch of color stuff
                        break;
        }
}
```

Temporal Cohesion

Elements are grouped into a module because they are all processed within the same limited time period

Common example

"Initialization" modules that provide default values for objects
"End of Job" modules that clean up

```
procedure initializeData()
        {
        font = "times";
        windowSize = "200,400";
        foo.name = "Not Set";
        foo.size = 12;
        foo.location = "/usr/local/lib/java";
        }
```

Cure – Each object should have a constructor and destructor

How is this better?

Procedural Cohesion

Associates processing elements on the basis of their procedural or algorithmic relationships

Procedural modules are application specific

In context the module seems reasonable

Removed from the context these modules seem strange and very hard to understand

Can not understand module without understanding the program and the conditions existing when module is called

Makes module hard to modify, understand

Class Builder verse Program writer

Communication Cohesion

Operations of a module all operate upon the same input data set and/or produce the same output data

Cure - Isolate each element into separate modules

Rarely occurs in object-oriented systems due to polymorphism (overloading)

Sequential Cohesion

Sequential association the type in which the output data from one processing element serve as input data for the next processing element

A module that performs multiple sequential functions where the sequential relationship among all of the functions is implied by the problems or application statement and where there is a data relationship among all of the functions

Cure – Decompose into smaller modules

Functional Cohesion

If the operations of a module can be collectively described as a single specific function in a coherent way, the module has functional cohesion

If not, the module has lower type of cohesion

In an object-oriented system:

Each operation in public interface of an object should be functional cohesive

Each object should represent a single cohesive concept

Informational Strength Cohesion

Myers states:

"The purpose of an informational-strength module is to hide some concept, data structure, or resource within a single module.

An informational-strength module has the following definition:

It contains multiple entry points

Each entry point performs a single specific function

All of the functions are related by a concept, data structure, or resource that is hidden within the module"

Object Cohesion

The degree to which components of a class are tied together

Evaluating cohesion requires:

- Technical knowledge of the application domain
- Some experience in building, modifying, maintaining, testing and managing applications in the appropriate domain
- Technical background in and experience with reusability

Questions to probe cohesiveness of an object

Does the object represent a complete and coherent concept or does it more closely resemble a partial concept, or a random collection of information?

Does the object directly correspond to a "real world entity," physical or logical?

Is the object characterized in very non-specific terms?

Collection of data, statistics, etc.

Do each of the methods in the public interface for the object perform a single coherent function?

If the object (or system of objects) is removed from the context of the immediate application, does it still represent a coherent and complete object-oriented concept?

Questions to probe cohesiveness of system of objects

Does the system represent an object-oriented concept?

Do all the objects directly support, or directly contribute to the support of, the object-oriented concept that the system represents?

Are there missing objects?

Objects in Isolation

Isolation means without considering any hierarchy that may contain the object or class

Individual Objects

A **primitive method** is any method that cannot be implemented simply, efficiently, and reliably without knowledge of the underlying implementation of the object

A **composite method** is any method constructed from two or more primitive methods – sometimes from different objects

A **sufficient set of primitive methods** for an object is a minimum set of primitive methods to accomplish all necessary work with on the object

A sufficient set of primitive methods has two major problems:

- Some tasks may be awkward and/or difficult with just a sufficient set of primitive methods

- A sufficient set of primitive methods may not allow us to fully capture the abstraction represented by the object

A **complete set of primitive methods** is a set of primitive methods that both allows us to easily work with the object, and fully captures the abstraction represented by the object.

To implement Java Collection

Subclass java.util.AbstractList and implement

- add(int index, Object element)
- get(int index)
- remove(int index)
- size()
- set(int index, Object element)

Subclass java.util.AbstractCollection and implement

- add(int index, Object element)
- iterator()
- size()

Iterator implements

- hasNext()
- next()
- remove()

Is either of these a sufficient set of primitive methods?

Java's ArrayList

add(int index, Object element)	add(Object o)	addAll(Collection c)
addAll(int index, Collection c)	clear()	clone()
contains(Object elem)	containsAll	**ensureCapacity**(int minCapacity)
equals	get(int index)	hashCode
indexOf(Object elem)	isEmpty()	iterator
lastIndexOf(Object elem)	listIterator	remove(int index)
removeAll	retainAll	set(int index, Object element)
size()	subList	toArray()
toArray(Object[] a)	toString	**trimToSize()**

Is this a complete set of primitive methods?

Ruby Array

-	&	*	[]	[]=	|
+	<<	<=>	==	abbrev	all?
any?	assoc	at	clear	collect	collect!
compact	compact!	concat	delete	delete_at	delete_if
detect	each	each_index	each_with_index	empty?	entries
eql?	fetch	fill	find	find_all	first
flatten	flatten!	frozen?	grep	hash	include?
index	indexes	indices	initialize_copy	inject	insert
inspect	join	last	length	map	map!
max	member?	min	nitems	pack	partition
pop	push	rassoc	reject	reject!	replace
reverse	reverse!	reverse_each	rindex	select	shift
size	slice	slice!	sort	sort!	sort_by
to_a	to_ary	to_s	to_set	transpose	uniq
uniq!	unshift	values_at	zip		

Smalltalk OrderedCollection 1

,	=	add:	add:after:	add:before:
add:beforeIndex:	addAll:	addAllFirst:	addAllLast:	addFirst:
addLast:	addLastNoCheck:	after:	allButFirst:	allButLast:
allSatisfy:	anySatisfy:	asArray	asBag	asFixedArgument
asList	asOrderedCollection	asSet	asSortedCollection	asSortedCollection:
asSortedStrings	asSortedStrings:	asSortedStrings:with:	asSortedStringsWith:	at:
at:put:	atAll:put:	atAllPut:	before:	capacity
changeCapacityTo:	changeSizeTo:	collect:	contains:	copyEmpty
copyEmpty:	copyFrom:to:	copyReplaceAll:with:	copyReplaceFrom:to:with:	copyUpTo:
copyWith:	copyWithout:	detect:	detect:ifNone:	do:
do:separatedBy:	doWithIndex:	emptyCheck	emptyCollectionError	errorOutOfBounds
find:	findFirst:	findFirst:startingAt:	findLast:	first
first:	firstObjectError	fold:	forStackDumpPrintUsing:	groupedBy:
grow	growSize	growToAtLeast:	hash	identityIndexOf:
includes:	identityIndexOf:ifAbsent:		identityIndexOf:from:to: ifAbsent:	

Smalltalk OrderedCollection 2

increaseCapacity	indexOf:	indexOf:ifAbsent:	inject:into:	insert:before:
inspectorClass	inspectorClasses	isEmpty	isNotEmpty	isSameSequenceAs:
isSequenceable	isWeakContainer	isWeakContainer:	keysAndValuesDo:	last
last:	lastIndexOf:	lastIndexOf:ifAbsent:	lastObjectError	literalArrayEncoding
makeRoomAtFirst	makeRoomAtLast	maxPrint	newReadWriteStream	nextIndexOf:from:to:
noMatchError	noSuchElementError	notEmpty	notEnoughElementsError	notFoundError
notKeyedError	occurrencesOf:	piecesCutWhere:	piecesCutWhere:do:	prevIndexOf:from:to:
writeStream	printOn:	readStream	readWriteStream	reject:
remove:	remove:ifAbsent:	removeAll:	removeAllSuchThat:	removeAtIndex:
removeFirst	removeFirst:	removeIndex:	removeLast	removeLast:
replaceAll:with:	replaceAll:with:from:to:	replaceFrom:to:with:	replaceFrom:to:with:startingAt:	representBinaryOn:
reverse	reverseDo:	runsFailing:	runsFailing:do:	runsSatisfying:
runsSatisfying:do:	select:	setIndices	setIndicesFrom:	size
storeOn:	swap:with:	tokensBasedOn:	trim	with:do:
			printBriefInspectorTextOn:	

Smalltalk OrderedCollection 3

decrementBy:boundedBy:highValue:wrapAround:
startingAt:replaceElementsIn:from:to:
replaceElementsFrom:to:withArray:startingAt:
replaceElementsFrom:to:withByteArray:startingAt:
replaceElementsFrom:to:withByteEncodedString:startingAt:
replaceElementsFrom:to:withCharacterArray:startingAt:
replaceElementsFrom:to:withIntegerArray:startingAt:
replaceElementsFrom:to:withLinkedList:startingAt:
replaceElementsFrom:to:withSequenceableCollection:startingAt:
replaceElementsFrom:to:withTwoByteString:startingAt:
replaceElementsFrom:to:withWordArray:startingAt:
indexOfSubCollection:startingAt:
indexOfSubCollection:startingAt:ifAbsent:
incrementBy:boundedBy:lowValue:wrapAround:

Levels of Cohesion

An object is not as cohesive as it could be if the public interface contains:

- Only primitive methods, but does not fully capture the abstraction represented by the object
- Primitive and composite methods, but does not fully capture the abstraction represented by the object
- A sufficient set of primitive methods with composite methods
- No primitive methods, just composite methods

Note

- Objects with a sufficient set of primitive methods with composite methods is more cohesive than objects with out a sufficient set of primitive methods
- All public methods must directly support the abstraction represented by the object. The methods must make sense when object is removed from the application

Composite Objects

A **composite object** is an object that is conceptually composed of two, or more, other objects, which are externally discernible.

Component objects are those that make up the composite object.

Component objects are **externally discernible** if

> The externally discernible state of the object is directly affected by the presence or absence of one or more component objects
>
> Component objects can be directly queried or changed via methods in the public interface of the composite object and/or

Ranking of Cohesion of Composite Objects
Increasing order of Goodness

Externally discernible component objects not related

Some externally discernible component objects are related, the group component objects does not make sense

The group component objects does not represent a single stable object-oriented concept, but are all bound together some how in an application

A majority of the externally discernible component objects support a single, coherent, object-oriented concept, but at least one does not

All of the externally discernible component objects support a single, coherent, object-oriented concept, but at least one needed is missing

All of the externally discernible component objects support a single, coherent, object-oriented concept, and none are missing

Accessing Cohesion of an Individual Object

Assessment of the public methods/public non-methods/component objects

Are all the items appropriate for the given object?

Do we have at least a minimally sufficient set of items?

Do we have extra or application-specific items?