SOFTENG 701: Advanced Software Engineering Development Methods Part 2

Lecture 4a: Coupling and Cohesion

Ewan Tempero
Department of Computer Science

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Agenda

- Admin
    - Assignment?
- Assignment 4
- Part 1
    - Coupling
    - Cohesion
- Part 2
    - The “CK” object-oriented metric suite for

Reading

- OO Design Quality Metrics, Robert Martin, 1994. please read for Tuesday (Available via **Reading Lists** on Canvas)
- E. Tempero, J. Noble, and H. Melton. “How do Java programs use inheritance? an empirical study of inheritance in Java software” *22nd European Conference on Object-Oriented Programming (ECOOP)*, 2008.
- See also `reading.pdf` — list of questions that might be useful for getting the most from reading a research paper

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Assignment 4

- Determine the empirical relationship of 6 designs for Kalah with respect to how good they are as object-oriented designs (i.e. ranking them).
- Designs come with implementations and measurements from many metrics.
- Write a report explaining your ranking, in particular how you determined it.

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Previously in SOFTENG701

. . .

ISO/IEC 25010 (Modularity) The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.*

Coupling and Cohesion

- Concepts to characterise the quality of “modules” based on our intuition of how systems can be built more easily
- Ideas that were developed in the 1960s†
- If can measure them, then can improve “design quality” (at least with respect to some notions of quality)

† First formally published as: W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974.

Definitions

Module a lexically contiguous sequence of program statements bounded by boundary elements, having an aggregate identifier†

Relationship A relationship exists between one module and another if that module cannot function correctly without the presence of the other

† Edward Yourdon and Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, 1979.

Definitions

Coupling: …the degree of interdependence between modules. …a measure of the *strength* of interconnection (Yourdon and Constantine, 1979)

- The more the connections between one module and the rest, the harder to understand that module, the harder to re-use that module in another situation, the harder it is to isolate failures caused by faults in the module
- $\Rightarrow$ The lower the coupling the “better”.

Cohesion: …the extent to which its individual components are needed to perform the same task. …how tightly bound or related [a module’s] internal elements are to one another (Yourdon and Constantine, 1979)

- The less tightly bound the internal elements, the more disparate the parts to the module, the harder it is to understand
- $\Rightarrow$ The higher the cohesion the “better”.

Classic coupling measurement

- “physical” concept
- ordinal categories of relationship between two modules

Content coupling modules can directly refer to the contents of each other “bad”

Common coupling modules communicate via global data

Control coupling modules communicate by data that allows one module to directly affect the behaviour of the other

Stamp coupling modules communicate by a heterogeneous set of items, not all of which are used

Data coupling modules communicate by parameters, where each parameter is a single item or a homogeneous set that incorporate no control element “good”

No coupling there is no relationship

Object-oriented coupling

- If classes are modules, what is object-oriented coupling?
- Recall definition

  Relationship A relationship exists between one module and another if that module cannot function correctly without the presence of the other

- Some classes in an implementation must have some relationships with other classes

⇒ there must be "necessary" coupling

⇒ "good design" includes removing (or avoiding) "unnecessary" coupling

- Question: how to tell necessary from unnecessary?

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

What coupling does this design have?

```
import java.util.Calendar;

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar dueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar)from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
import java.util.Calendar; Must be on classpath

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar dueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar)from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```

- coupling?
- interconnection?
- dependency?

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
import java.util.Calendar;

                      Must be an interface in default package
public class AdultIssuePolicy implements IssuePolicy {

    public Calendar dueDate(BiblioType type, Calendar from) {

        Calendar result = (Calendar)from.clone();

        result.add(Calendar.DATE, 14);

        return result;

    }

}
```

- type relationship
- can we understand `AdultIssuePolicy` without considering `IssuePolicy`?
- can we reuse `AdultIssuePolicy` without requiring `IssuePolicy`?
- Can we isolate faults in `AdultIssuePolicy` (or `IssuePolicy`) from `IssuePolicy` (or `AdultIssuePolicy`)?

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
import java.util.Calendar;


public class AdultIssuePolicy implements IssuePolicy {
                             Must be in default package
    public Calendar dueDate(BiblioType type, Calendar from) {

        Calendar result = (Calendar)from.clone();

        result.add(Calendar.DATE, 14);

        return result;

    }

}
```

- type relationship

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
import java.util.Calendar;


public class AdultIssuePolicy implements IssuePolicy {

    public Calendar dueDate(BiblioType type, Calendar from) {

        Calendar result = (Calendar)from.clone();
               Must be a method in Calendar (or ancestor) and
               must have 2 arguments of the expected types
        result.add(Calendar.DATE, 14);

        return result;

    }

}
```

- Do we need to understand something of `Calendar` in order to understand `AdultIssuePolicy`?
- Can `AdultIssuePolicy` be reused independent of `Calendar`?
- Can a fault in `Calendar` result in a failure in `AdultIssuePolicy` (and vice versa)?

Example

```
import java.util.Calendar;


public class AdultIssuePolicy implements IssuePolicy {

    public Calendar dueDate(BiblioType type, Calendar from) {

        Calendar result = (Calendar)from.clone();
                   Must be of the correct type
        result.add(Calendar.DATE, 14);

        return result;

    }

}
```

- Can a fault in Calendar result in a failure in AdultIssuePolicy (and vice versa)?
- Is it a different fault to previous example?

Example

```
import java.util.Calendar;


public class AdultIssuePolicy implements IssuePolicy {

    public Calendar dueDate(BiblioType type, Calendar from) {
                                      Must implement Cloneable
        Calendar result = (Calendar)from.clone();

        result.add(Calendar.DATE, 14);

        return result;

    }

}
```

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

What coupling is unnecessary?

```
import java.util.Calendar;

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar dueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar)from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

What coupling is unnecessary?

```
import java.util.Calendar;

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar dueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar)from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```

What coupling is unnecessary?

```
import java.util.Calendar;

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar dueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar)from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```

- unnecessary?
- coupling?
- type relationship

Object-oriented coupling

- What “dependencies” should we consider to be “coupling”?
- What is the “strength” of the coupling?
  - ○ number of occurrences?
  - ○ “kind” of coupling?
  - ○ “location” of coupling? (parameter vs. field vs. local)

Example

```
import java.util.Calendar;

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar dueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar)from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```

- “strength” for `Calendar` = 5? 4? 1?

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
import java.util.Calendar;

public class AdultIssuePolicy implements IssuePolicy {
    public Calendar dueDate(BiblioType type, Calendar from) {
        Calendar result = (Calendar)from.clone();
        result.add(Calendar.DATE, 14);
        return result;
    }
}
```

- more strongly coupled with `IssuePolicy` or `BiblioType`?

Coupling FAQs

- If A collaborates with B is A coupled to B?
- If A is coupled to B is B coupled to A?
- If A invokes a method of B but C only uses B as the type of a parameter, is A "more coupled" than C to B?
- If A calls one method on B and C calls 5 methods on B, is C "more coupled" than A to B?
- If A inherits from B is A coupled to B?
  - what about if A overrides all of B's methods?
  - what about if A overrides some of B's methods?
  - what about if A makes a self-call on one of B's methods?
  - What about if A refers to a field in B?
  - . . . and which of these is "more coupled"?
- If A uses a getter method on B and C uses a "proper" method is A "more coupled" than C to B?
- . . . and many many more

“Classic” cohesion metric

- “logical” concept
- ordinal scale for one module

Coincidental module performs unrelated functions

Logical module performs functions that are related only logically

Temporal module performs more than one function, but they all happen within a well-defined timespan

Procedural module performs more than one function, but they all do functions that are related

Communicational module performs more than one function, but they are all on the same body of data

Sequential module performs more than one function, but they occur in a well-defined (specified) order

Functional module performs exactly one easily identifiable function

Informational module performs more one independent function with single entry and exit points operating on the same body of data

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
public class Utility {
  public String removeSpaces(String str) {
    ....
  }

  public int power(int a, int b) {
    ....
  }
}
```

- Not obvious relationship between the two methods, just placed together for convenience

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
public class Initialisation {
  public void openFiles(File[] files) {
    ....
  }

  public void initArrays(int[][] array1, String[] array2) {
    ....
  }
}
```

- Relationship between `openFiles` and `initArrays` is mainly one of “intent” (initialisation) rather than a concept
- `openFiles` and `initArrays` do not work together to provide the intent

Example

```
public class ChartDrawer {
  private Screen screen;
  private Data data;
  public void clearScreen() { .... }
  public void readData(InputSource source) { .... }
  public void drawAxes() { .... }
  public void drawLabel(String label) { .... }
  public void drawChart() { .... }
  public void print(Printer p) { .... }
}
```

- While all members work towards performing one task, they deal with different abstractions
  - creating the chart
  - changing what's on a screen
  - managing data
  - printing
- Question: if the screen implementation changes, what else needs to change?

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
public class LineManager {
  public LineManager(Point end1, Point end2) {
    // store coordinates
  }

  public void draw(Window w) {
    // Draw line on w
  }

  public float computeLength() {
    ....
  }
}
```

- Concept is more well defined than previous examples
- Drawing is different to geometric computation (length)

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

```
public class Chart {
  public void readInput(InputSource source) {
    ....
  }

  public void cleanData() {
    ....
  }

  public void filterData(Filter f) {
    ....
  }

  public void createChart() {
    ....
  }
}
```

- Seems like one concept
- Managing data is different from drawing a chart

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example

From the Java Standard Library:

```
public class Stack extends Vector {
   Stack() { ... }
   boolean empty() { ... }
   Object peek() { ... }
   Object pop() { ... }
   Object push(Object item) { ... }
   int search(Object o) { ... }
}
```

- A Stack is not a Vector — mixing abstractions
- In the “standard” definition of Stack, searching is not considered
- “. . . the extent to which its individual components are needed to perform the same task.”

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example: Cohesion

```
public class PersonDetails {
  private String _firstname;
  private String _surname;
  private String _street;
  private String _city;
  public PersonDetails() {}
  public setName(String f, String s) {
      _firstname = f; _surname = s;
  }
  public setAddress(String st, String c) {
      _street = st; _city = c;
  }
  public void printAddress() {
      System.out.println(_street);
      System.out.println(_city);
  }
  public void printName() {
      System.out.println(_firstname + " " + _surname);
  }
}
```

- “. . . how tightly bound or related [a module’s] internal elements are to one another”

- Agenda
- Assignment 4
- Review
- Coupling and Cohesion
- Classic Coupling
- OO Coupling
- Measuring Coupling
- Classic Cohesion
- (Non) Examples
- Cohesion
- Evaluation
- Key Points

Example: Cohesion

```
public class PersonDetails {
  private String _firstname;
  private String _surname;
  private String _street;
  private String _city;
  public PersonDetails() {}
  public setName(String f, String s) {
      _firstname = f; _surname = s;
  }
  public setAddress(String st, String c) {
      _street = st; _city = c;
  }
  public void printAddress() {
      System.out.println(_street);
      System.out.println(_city);
  }
  public void printName() {
      System.out.println(_firstname + " " + _surname);
  }
}
```

- “…how tightly bound or related [a module’s] internal elements are to one another”

Questions to consider

- What quality attributes are consider related to coupling and cohesion?
- What is the relationship between those quality attributes and coupling and cohesion?

Key Points

- Coupling and cohesion are attributes of design quality that have been around a long time
- The intuition supporting them is appealing
- They are constructs, and so expect there to be difficulty in measuring them