

Refactoring

Lecture 5:

Metrics

DAT159/H18
Volker Stolz

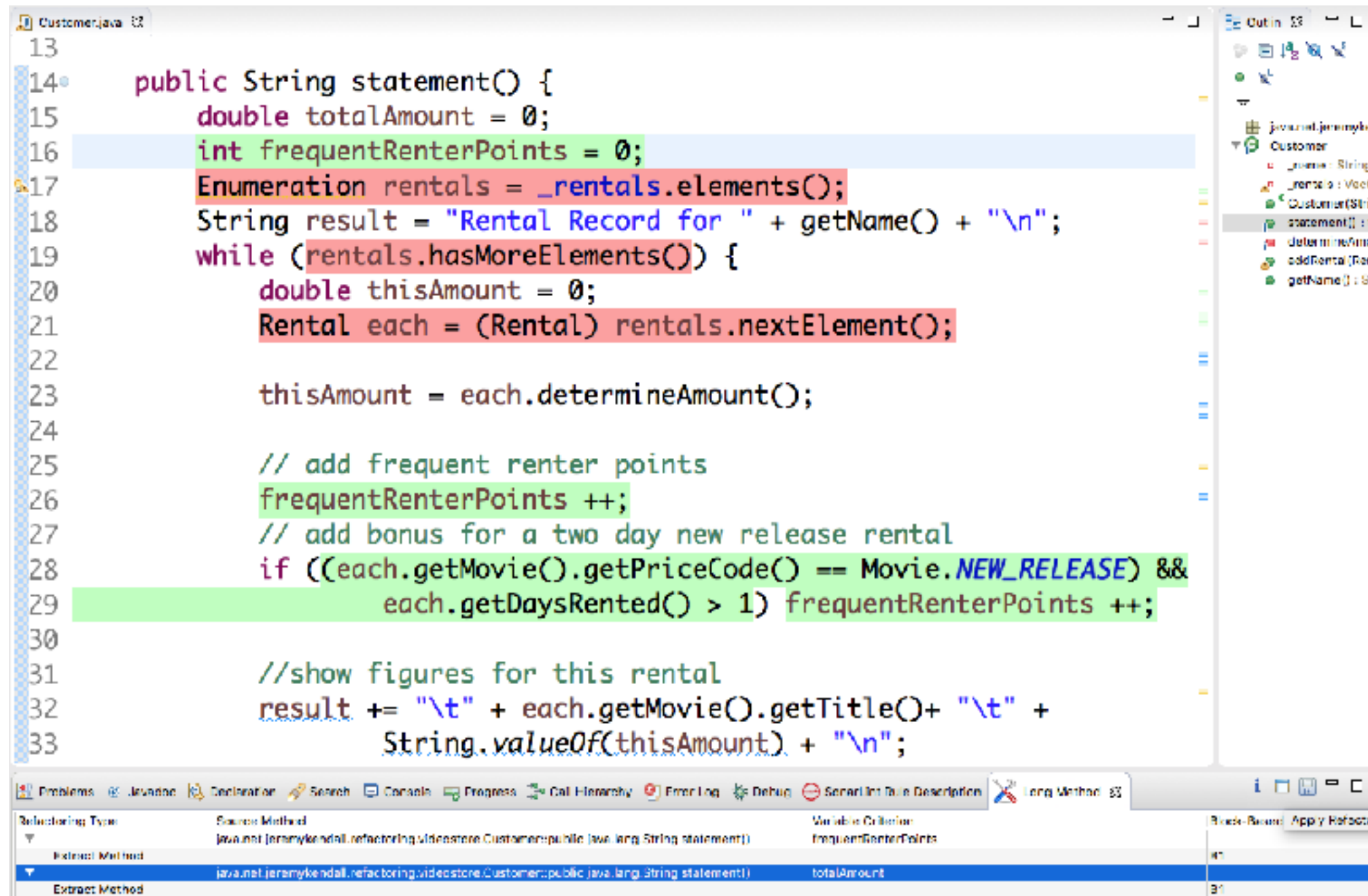
Supported by the bilateral SIU/CAPES
project “Modern Refactoring” 2017/18



Høgskulen
på Vestlandet

Smells Addendum

JDeodorant: <https://marketplace.eclipse.org/content/jdeodorant>



```
13
14 public String statement() {
15     double totalAmount = 0;
16     int frequentRenterPoints = 0;
17     Enumeration rentals = _rentals.elements();
18     String result = "Rental Record for " + getName() + "\n";
19     while (rentals.hasMoreElements()) {
20         double thisAmount = 0;
21         Rental each = (Rental) rentals.nextElement();
22
23         thisAmount = each.determineAmount();
24
25         // add frequent renter points
26         frequentRenterPoints++;
27         // add bonus for a two day new release rental
28         if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
29             each.getDaysRented() > 1) frequentRenterPoints++;
30
31         //show figures for this rental
32         result += "\t" + each.getMovie().getTitle() + "\t" +
33             String.valueOf(thisAmount) + "\n";
34     }
35     result += "\nTotal Amount: " + result + "\n";
36     return result;
37 }
```

Refactoring Type	Source Method	Variable Criterion	Back-Bound	Apply Refactor
Extract Method	java.net.jeremykendall.refactoring.videostore.Customer:public java.lang.String statement()	frequentRenterPoints	47	
Extract Method	java.net.jeremykendall.refactoring.videostore.Customer:public java.lang.String statement()	totalAmount	31	

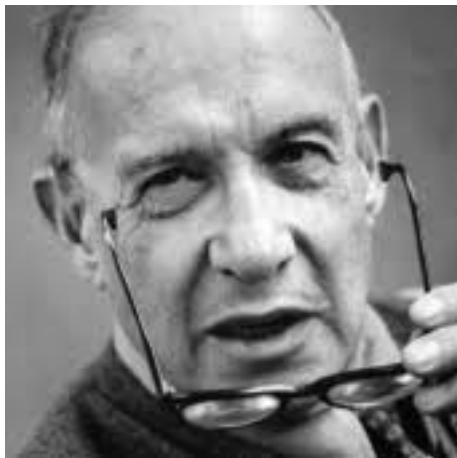
JDeodorant: Feature Envy

The screenshot shows the Eclipse IDE with the JDeodorant Feature Envy tool. The main editor displays the `printNotification` method in `FastLeaderElection.java`. The method is highlighted in green, and the Feature Envy tool window at the bottom lists 12 references to the method.

```
692 }
693 }
694
695 private void printNotification(Notification n){
696     LOG.info("Notification: "
697         + Long.toHexString(n.version) + " (message format vers
698         + n.leader + " (n.leader), 0x"
699         + Long.toHexString(n.zxid) + " (n.zxid), 0x"
700         + Long.toHexString(n.electionEpoch) + " (n.round), " +
701         + " (n.state), " + n.sid + " (n.sid), 0x"
702         + Long.toHexString(n.peerEpoch) + " (n.peerEpoch), "
703         + self.getPeerState() + " (my state)"
704         + (n.qv!=null ? (Long.toHexString(n.qv.getVersion())) +
705     }
706
707
708 /**
709  * Check if a pair (server id, zxid) succeeds our
710  * current vote.
711  *
712  * @param id    Server identifier
```

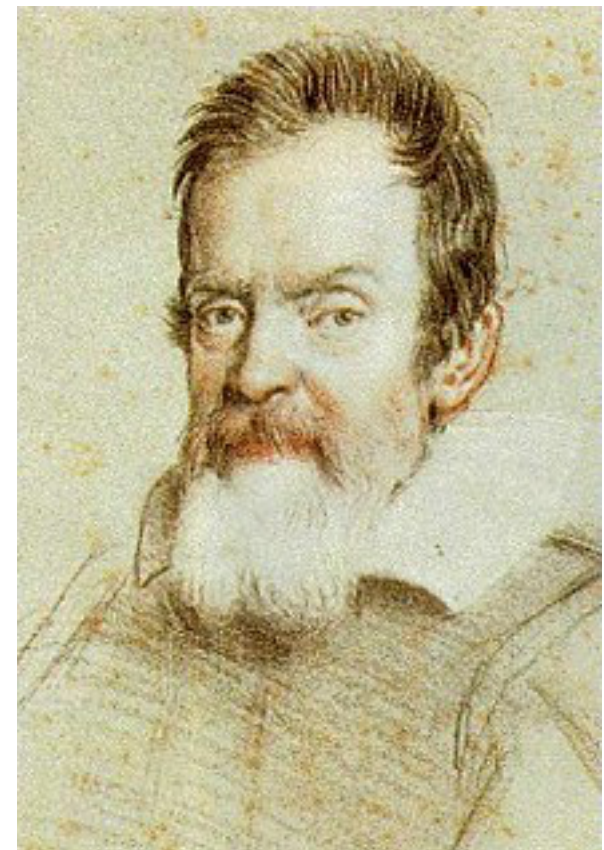
Referencing Type	Source Entity	Target Class	Source/Target accessed members	Rule ID
Move Method	org.apache.zookeeper.server.quorum.FastLeaderElection	org.apache.zookeeper.server.quorum.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	
Move Method	org.apache.zookeeper.server.FastLeaderElection	org.apache.zookeeper.server.FastLeaderElection	1/2	

Metrics



“If you can't measure it, you can't improve it.” -
Peter Drucker on management

“Measure what is measurable, and
make measurable what is not so”
- Galileo(?)



Metrics for Software Engineering?

Code metrics:

- (S)LOC
- Cyclomatic Complexity
- Depth of Inheritance
- Cohesion
- Coupling

**Object-
oriented**

Other metrics:

- Performance
- Test coverage
- Bugs (per LOC)
- Function Point Analysis

SLOC Measurement

- SLOC is the traditional and the most popular sizing metric
- Excludes comments and blanks
- Includes headers, declarations,...
- Counts individual lines; doesn't care about multiple statements/line
- LLOC: *logical* LOC (statements only)
- Boehm: *delivered source instructions* (DSI)
- Q: How do the refactorings affect this metric?

SLOC vs. LLOC

Example:

- SLOC: 5
- LLOC: 2
(for, printf)
- What about ++?
- Neither cares about structure...

```
for (i = 0;  
    i < 100;  
    i++) {  
    printf("hello");  
}  
  
/* An important loop */
```

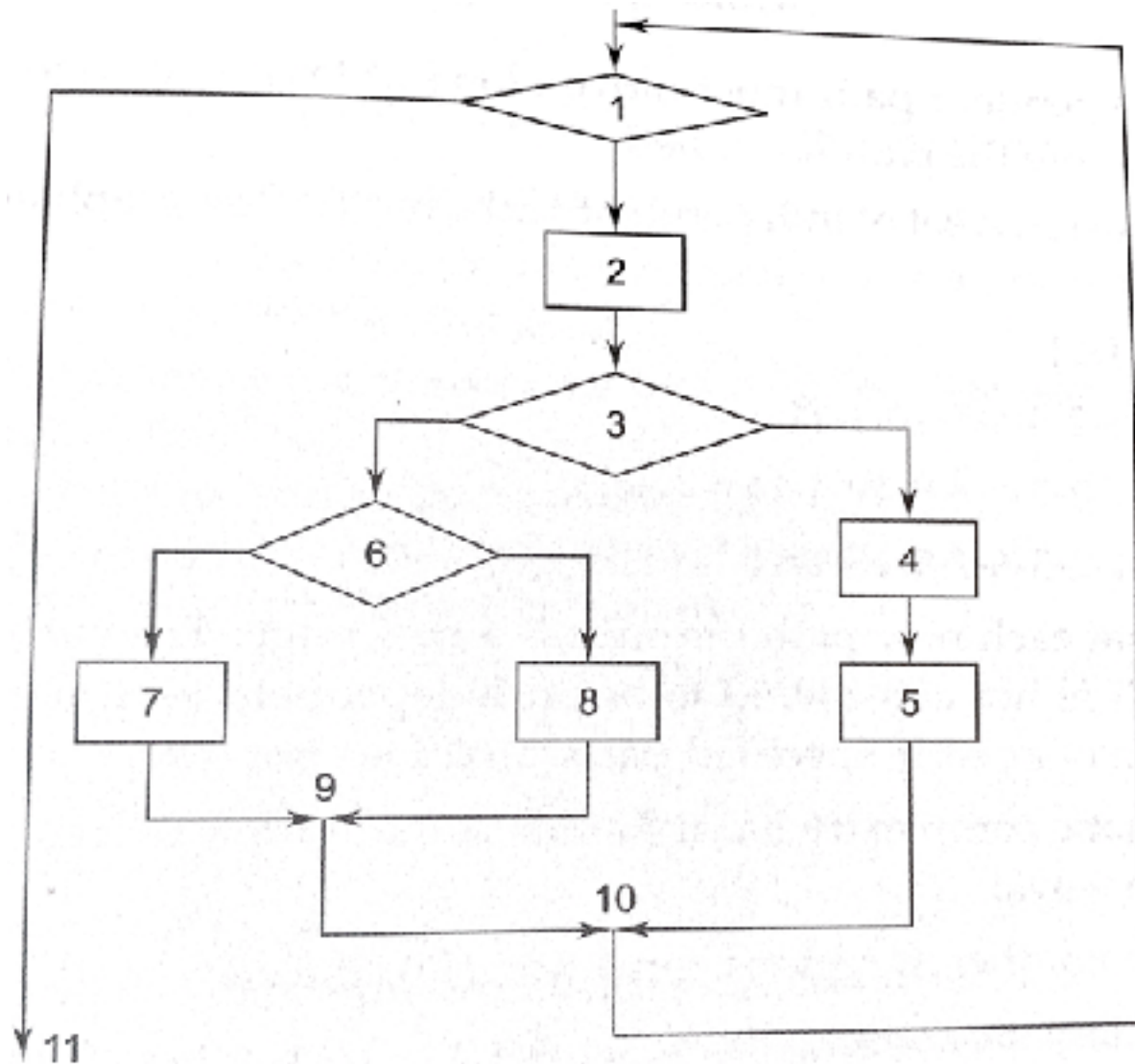
McCabe's Cyclomatic Complexity (MCC)

“Program Control Graph” G (now Control Flow Graph):

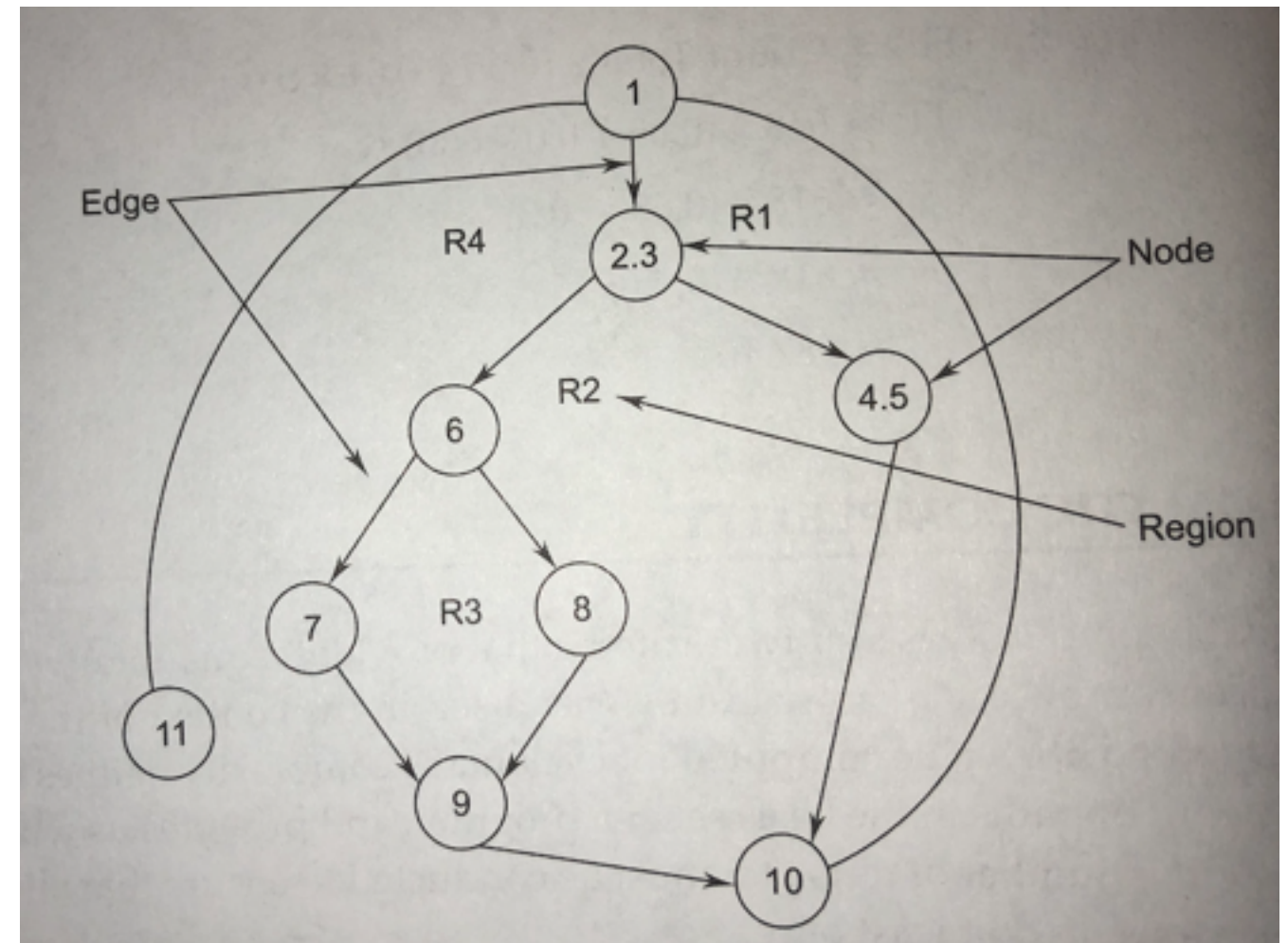
- node = block of code without jumps (N)
(*straight-line code, basic block*)
- edge = branches (E)
- connected components (p)
- $V(G) = E - N + p$ (variation: $+2p$; p usually 1, hence: $+2$)
- Relation to number of *paths* to tests

MAT101!

MCC: Flowchart vs. Flowgraph



Flowchart



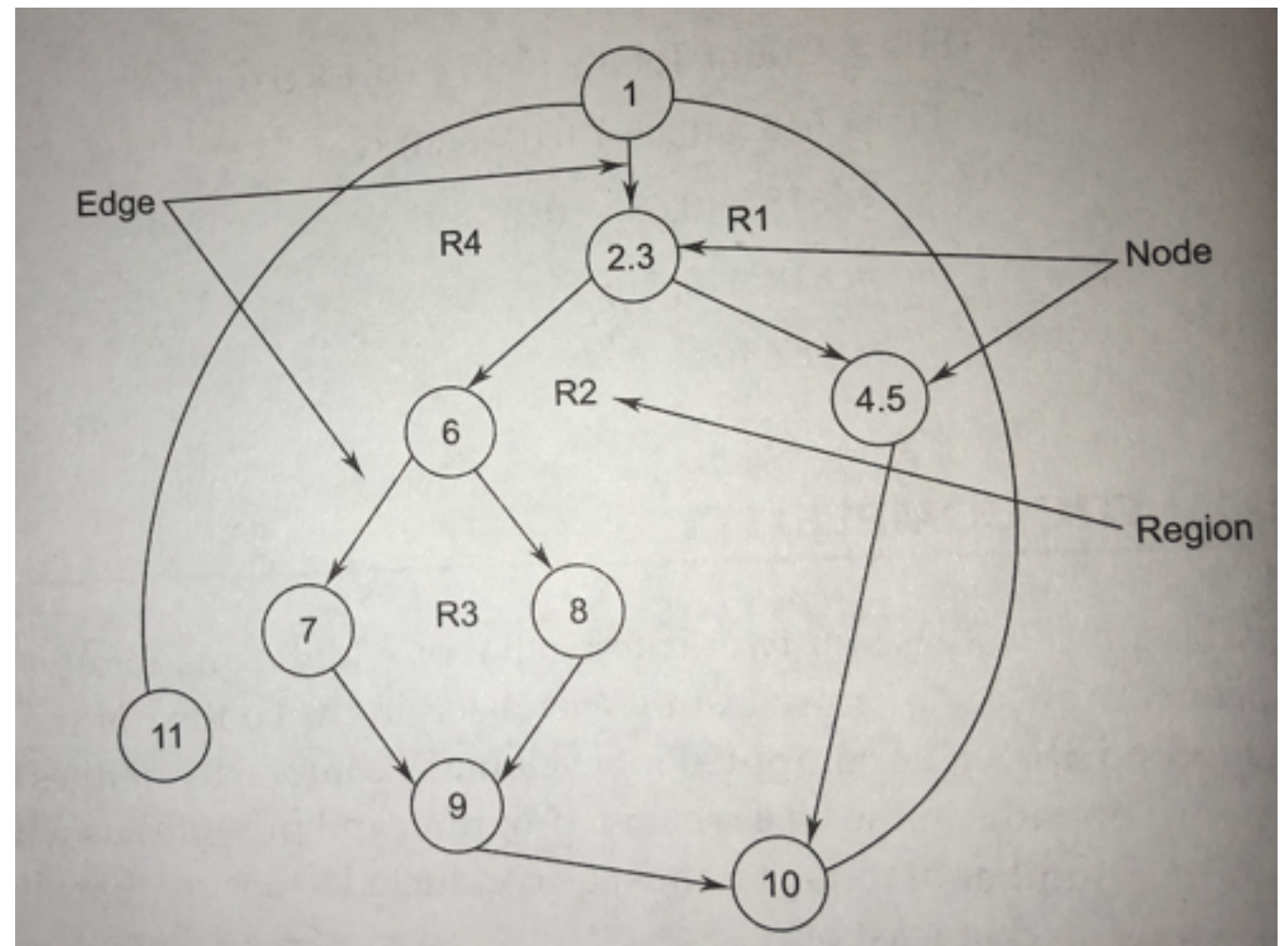
Flowgraph

[Agarwal, Tayal, Gupta: "Software Engineering & Testing"]

MCC Example

How many independent paths?

- Set of *independent* paths:
 - 1-11
 - 1-2-3-4-5-10-1-11
 - 1-2-3-6-8-9-10-1-11
 - 1-2-3-6-7-9-10-1-11
- **$V(G) = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4 \text{ (regions)}$**
Alternative: $V(G) = 3 \text{ predicate nodes} + 1 = 4$



Control Flow Graphs

Exercise — draw CFGs for Java's constructs:

- `if-then-else`
- `while () {}`
- `do {} while ()`
- `for () {}`

(Bonus question: what about exceptions?)

Cyclomatic Complexity: Example

```
public static void bubbleSort(int[] numArray) {  
  
    int n = numArray.length;  
    int temp = 0;  
  
    for (int i = 0; i < n; i++) {  
        for (int j = 1; j < (n - i); j++) {  
  
            if (numArray[j - 1] > numArray[j]) {  
                temp = numArray[j - 1];  
                numArray[j - 1] = numArray[j];  
                numArray[j] = temp;  
            }  
  
        }  
  
    }  
}
```

```
int partition(int arr[], int left, int right)  
{  
    int i = left, j = right;  
    int tmp;  
    int pivot = arr[(left + right) / 2];  
  
    while (i <= j) {  
        while (arr[i] < pivot)  
            i++;  
        while (arr[j] > pivot)  
            j--;  
        if (i <= j) {  
            tmp = arr[i];  
            arr[i] = arr[j];  
            arr[j] = tmp;  
            i++;  
            j--;  
        }  
    };  
    return i;  
}
```

```
void quickSort(int arr[], int left, int right) {  
    int index = partition(arr, left, right);  
    if (left < index - 1)  
        quickSort(arr, left, index - 1);  
    if (index < right)  
        quickSort(arr, index, right);  
}
```

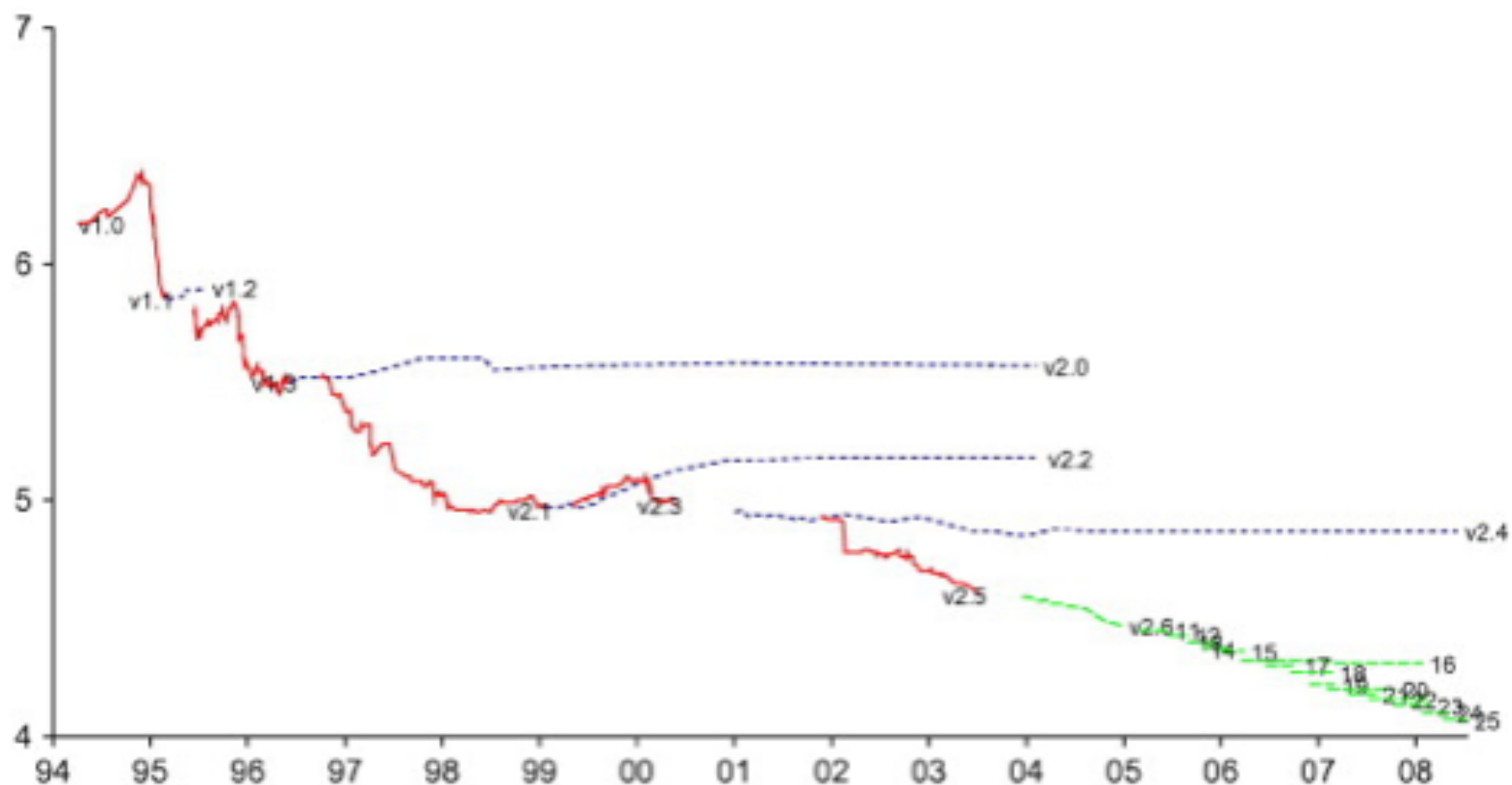
Cyclomatic Complexity (3)

- Independent of number of lines per function
(number of blocks and branches, not size of a block)
- Does not depend on format/coding style
- Let's ignore p for now!
- Reasonable values? 5? 10? More than 10?
- Q: How do the refactorings affect this metric?
- Applies to C and Java (no OO)
- Compare with Sonar's *Cognitive Complexity*

MCC in the Linux Kernel

Average cyclomatic complexity per function over time

Ayelet Israeli, Dror G. Feitelson: The Linux kernel as a case study in software evolution. Journal of Systems and Software 83(3): 485-501 (2010)



Depth of Inheritance

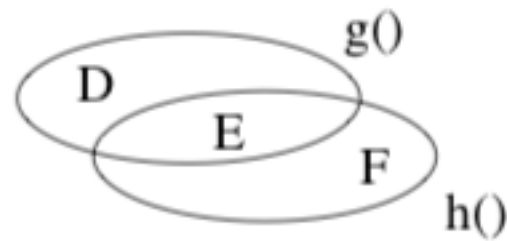
- Obvious [Chidamber & Kemerer], distance from superclass
- Effect on program understanding?
“The deeper a class [..], the greater the number of methods it is likely to inherit”
- Also: breadth of tree (“*top-heavy*”/“*bottom-heavy*”; former can indicate lack of reuse)

Lack of Cohesion

- Cohesion promotes encapsulation
- Lack of cohesion can recommend splitting classes
- LCOM1-4 (LCOM5?)
 - LCOM1 [Chidamber & Kemerer]
Number of pairs of methods that **do not share** attributes (higher = worse) — getters/setters?
 - LCOM4 [Hitz & Montazeri 1995]: number of "connected components" in a class. A connected component is a set of related methods (and class-level variables). There should be only one such a component in each class. If there are 2 or more components, the class should be split [..]
- Maybe you want to *Extract Method* first?

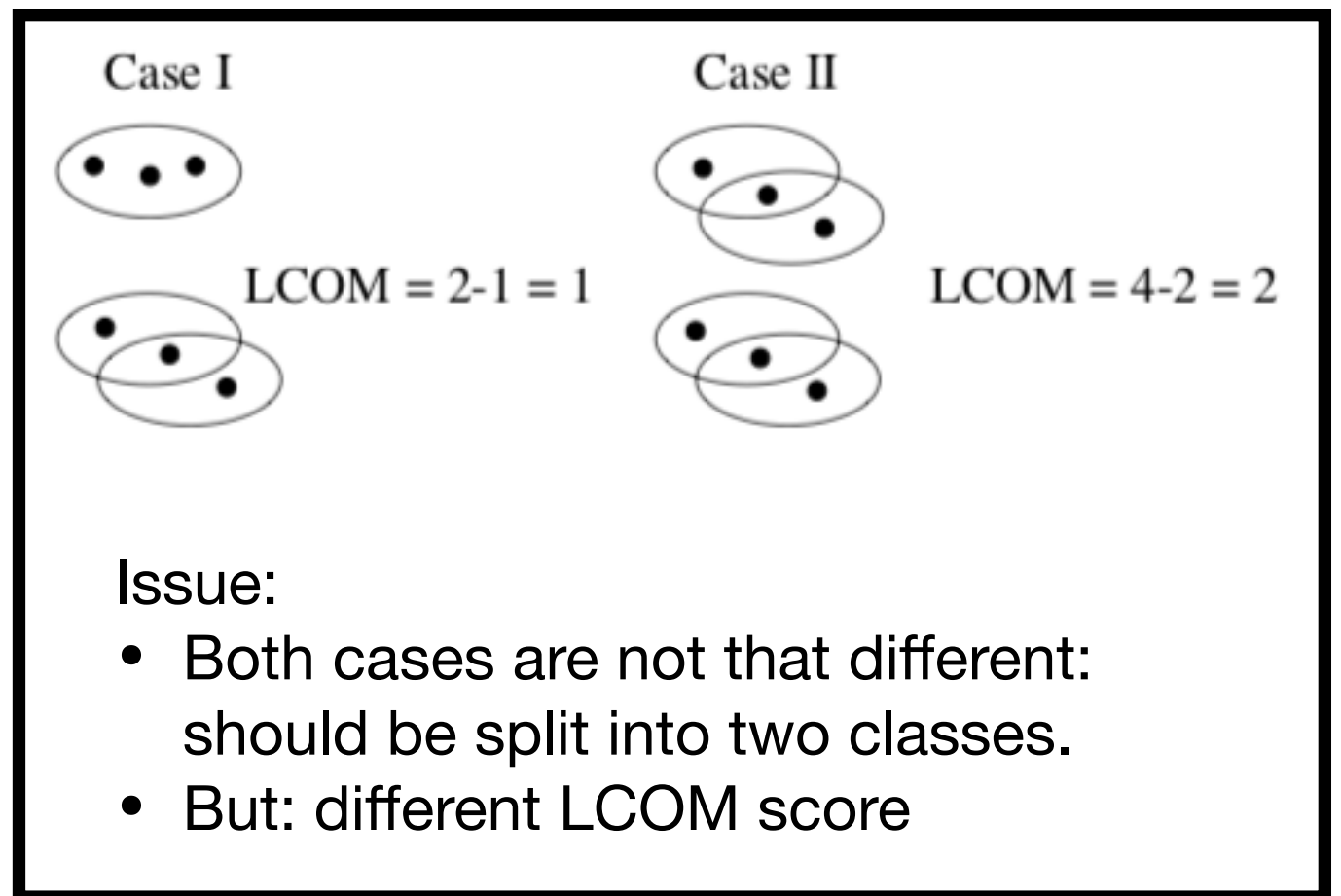
MAT101!!

LCOM Example



```
class X {  
    int A, B, C, D, E, F;  
    void f () { ... uses A, B, C ... }  
    void g() { ... uses D, E ... }  
    void h() { ... uses E, F ... }  
};
```

[Hitz/Montazeri]



Coupling/CBO

- **Coupling Between Object** classes [Chidamber & Kemerer]
- Number of other classes to which a class is coupled
- Objects *coupled*, if methods of one use methods or attributes of another
- High coupling = bad for reuse

Example CBO (1)

- CBO for classes A/B/C?
- What about $A \rightarrow B \rightarrow C$, is CBO associative? Should it be?

```
1 package dat159.metrics;
2
3 public class CBO {
4
5     class A {
6         B b;
7     }
8
9     class B {
10        A a1;
11        A a2;
12        C c;
13    }
14
15    class C {}
16 }
```

Example CBO (2)

```
1 package dat159.metrics;
2
3 public class CBO {
4
5     class A {
6         B b;
7     }
8
9     class B {
10        A a1;
11        A a2;
12        C c;
13    }
14
15    class C {}
16 }
```

```
5     class A {
6         B b;
7
8         void f() {
9             b.c.x = 42;
10        }
```

?

Tools for Metrics

- Install *Metrics* plugin from
 - ~~Eclipse Marketplace~~
 - ~~State-of-flow update site~~
 - `https://github.com/qxo/eclipse-metrics-plugin/raw/master/updatesite/`

References:

- Thomas J. McCabe:
“A Complexity Measure”. IEEE Trans. Software Eng. 2(4):
308-320 (1976)
- Shyam R. Chidamber, Chris F. Kemerer:
“A Metrics Suite for Object Oriented Design”. IEEE Trans.
Software Eng. 20(6): 476-493 (1994)
- Agarwal, Tayal, Gupta: “Software Engineering & Testing”,
Jones and Bartlett Publishers, 2010, Ch. “Software
Measurement and Metrics”