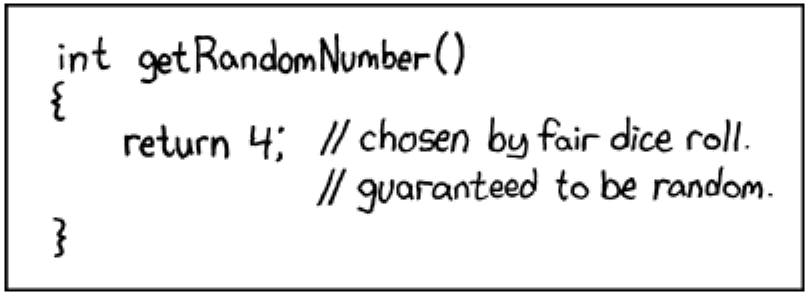


```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

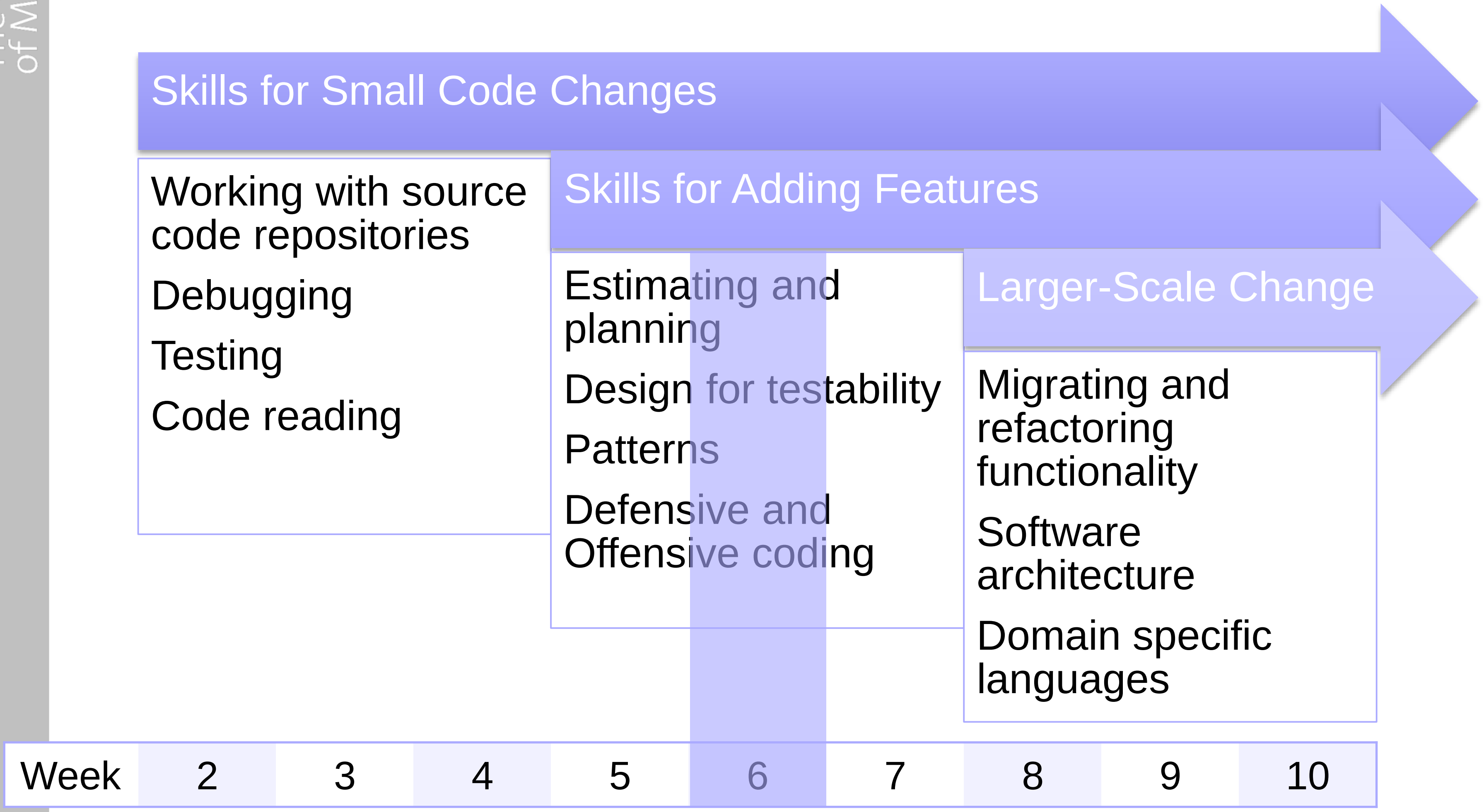
Design for Testability

COMP23420: Software Engineering

Week 6

Robert Haines and Caroline Jay

Course Unit Roadmap (Weeks 2-10)


Skills for Small Code Changes
Working with source code repositories
Debugging
Testing
Code reading
Skills for Adding Features
Estimating and planning
Design for testability
Patterns
Defensive and Offensive coding
Larger-Scale Change
Migrating and refactoring functionality
Software architecture
Domain specific languages
Week
2
3
4
5
6
7
8
9
10

Link to the Coursework/Exam

- We will learn:
  - Why it is important to think about testing when you design your code
  - Some simple techniques to improve testability of your code and existing code
    - There are many other techniques – this is just a taster!

- Coursework: Testing your new features will be easier if you consider testability as you go along

- Exam: There are questions about testing and testability in the exam

Design for Testability?

- What is Design for Testability?
  - Making sure that we can test what we build
- Why Design for Testability?
  - So that we can test what we build in *isolation*
- What prevents testability
  - Complexity
  - Non-deterministic code (different every time)
  - Hard coding implementation in the wrong place
  - Not allowing inheritance
  - Breaking the Law of Demeter

MANCHESTER
1824

The University of Manchester

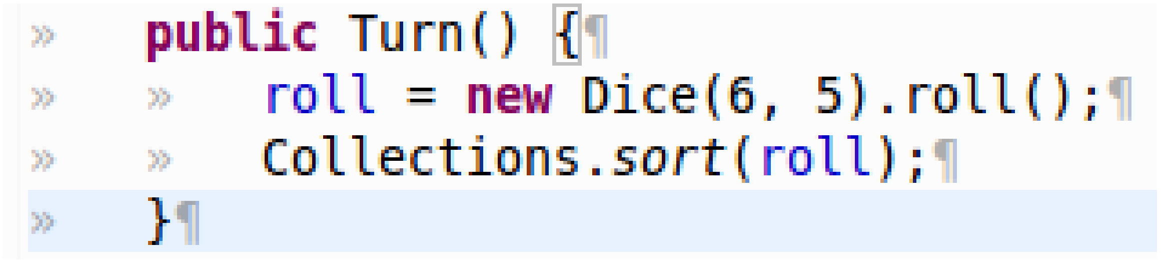
School of Computer Science

Design for testability exercise

- Introduction and exercise 1
- What have we learnt?

Be careful of new

- Avoid new until you really have to use it
  - The most common form of hardcoding
  - Nails down the *exact* implementation of an object
- Methods should only instantiate objects we don't want to substitute
  - Don't do this:



```
    public Turn() {
        roll = new Dice(6, 5).roll();
        Collections.sort(roll);
    }
```

Design for testability exercise

- Refactoring the Turn class
- Overriding methods to help testability
- Exercise 2
- What have we learnt?

Test doubles

- "Pretend" objects used in place of real objects for testing purposes

- Dummy objects
  - Passed around but never actually used. Often used to fill parameter lists
- Fake objects
  - Have working implementations. Usually take shortcuts which makes them unsuitable for use in production
- Stubs
  - Provide canned answers to calls made during the test, usually not responding anything outside the test
- Mocks
  - Pre-programmed with expectations which form a specification of the calls they are expected to receive

Mocks

- When “mocking” we create a special subclass of something to help us test something else

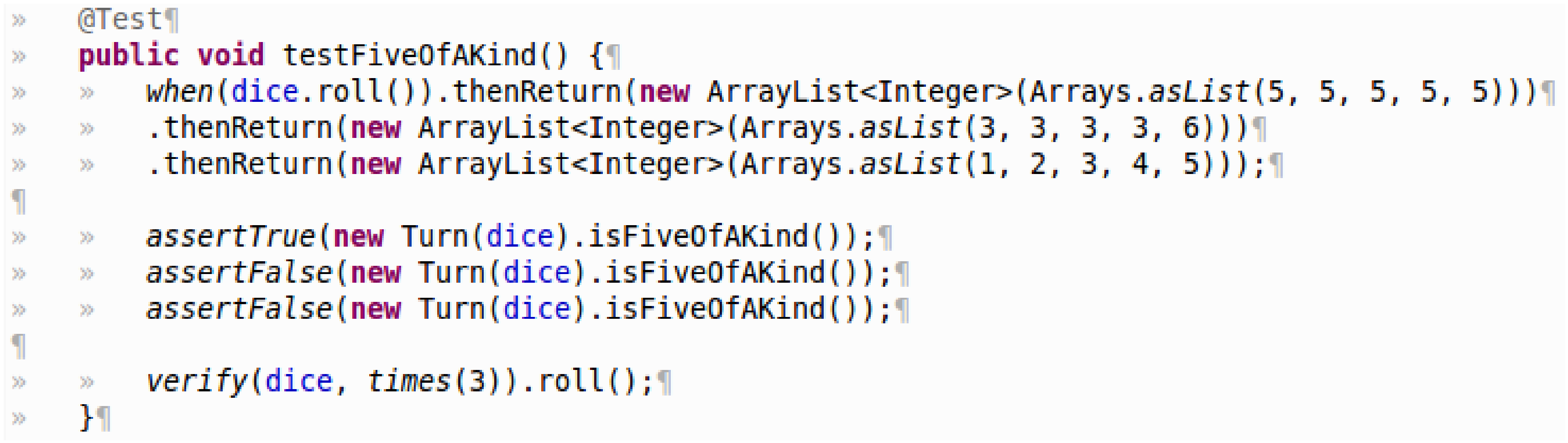
- With this subclass (a mock) we can
  - Control certain aspects of a class’s behaviour
    - Fix return values
  - Verify that certain behaviours occur
    - Methods called the correct number of times

Design for testability exercise

- Mocking classes for deeper testing
- Exercise 3
- Exercise 4
- Were there any problems with the methods in the Turn class?

Verifying behaviour

- With our mocks we can also verify how many times a method is called
  - Sometimes this is important
- Using our current example:

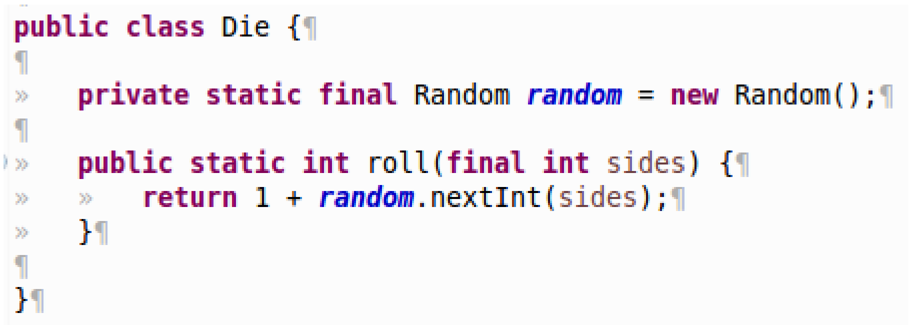


```
@Test
public void testFiveOfAKind() {
    when(dice.roll()).thenReturn(new ArrayList<Integer>(Arrays.asList(5, 5, 5, 5, 5)))
    .thenReturn(new ArrayList<Integer>(Arrays.asList(3, 3, 3, 3, 6)))
    .thenReturn(new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5)));

    assertTrue(new Turn(dice).isFiveOfAKind());
    assertFalse(new Turn(dice).isFiveOfAKind());
    assertFalse(new Turn(dice).isFiveOfAKind());

    verify(dice, times(3)).roll();
}
```

- Try different values for the `times()` method.

Other “gotchas” in the Five Dice code

- Avoid `static` method where you might need to replace things with a test dummy
  - `static` methods cannot be overridden (in Java)
- How would you mock the `Die` class?
  - Change the Random number generator?
  - Verify how many times roll was called for each Turn?



```
public class Die {

    private static final Random random = new Random();

    public static int roll(final int sides) {
        return 1 + random.nextInt(sides);
    }

}
```

- You can’t! So refactor away the `statics`.

Next Week

- In the team study sessions you will work on the coursework
- In the workshop we will learn about patterns and coding styles

Marauroa

- Have a look at the I18N class in the marauroa.common.i18n package
  - According to the report you did in Week 2 this has 53% coverage
  - But there are no tests!

- How does this class report as 53% covered by tests?
- Why might there be no tests for this class?