Testing

ENGI 5895: Software Design

Andrew Vardy

Faculty of Engineering & Applied Science
Memorial University of Newfoundland

March 6, 2017

Outline

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:
  - Test an individual unit of software (methods or complete classes)

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:
  - Test an individual unit of software (methods or complete classes)
- Integration Testing:

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:
  - Test an individual unit of software (methods or complete classes)
- Integration Testing:
  - Individual software components are combined and tested as a group

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:
  - Test an individual unit of software (methods or complete classes)
- Integration Testing:
  - Individual software components are combined and tested as a group
- System Testing:

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:
  - Test an individual unit of software (methods or complete classes)
- Integration Testing:
  - Individual software components are combined and tested as a group
- System Testing:
  - The system as a whole is tested

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:
  - Test an individual unit of software (methods or complete classes)
- Integration Testing:
  - Individual software components are combined and tested as a group
- System Testing:
  - The system as a whole is tested

Levels of Testing

"Program testing can be used to show the presence of bugs, but never to show their absence!" [E. W. Dijkstra]

- Unit Testing:
  - Test an individual unit of software (methods or complete classes)
- Integration Testing:
  - Individual software components are combined and tested as a group
- System Testing:
  - The system as a whole is tested

All of these are important, but methods for integration and system testing will depend on your application. We focus here on **unit testing**.

Testing Methods

- White Box Testing:

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria
    - e.g Code coverage: Writing tests to ensure that all program statements are executed at least once)

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria
    - e.g Code coverage: Writing tests to ensure that all program statements are executed at least once)
- Black Box Testing:

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria
    - e.g Code coverage: Writing tests to ensure that all program statements are executed at least once)
- Black Box Testing:
  - The tester has no access to the underlying implementation, but focusses instead upon testing the system to verify that the functional requirements have been met

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria
    - e.g Code coverage: Writing tests to ensure that all program statements are executed at least once)
- Black Box Testing:
  - The tester has no access to the underlying implementation, but focusses instead upon testing the system to verify that the functional requirements have been met
  - Advantage: Tester is impartial

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria
    - e.g Code coverage: Writing tests to ensure that all program statements are executed at least once)
- Black Box Testing:
  - The tester has no access to the underlying implementation, but focusses instead upon testing the system to verify that the functional requirements have been met
  - Advantage: Tester is impartial
  - Disadvantage: Tester may not exercise all parts of the code

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria
    - e.g Code coverage: Writing tests to ensure that all program statements are executed at least once)
- Black Box Testing:
  - The tester has no access to the underlying implementation, but focusses instead upon testing the system to verify that the functional requirements have been met
  - Advantage: Tester is impartial
  - Disadvantage: Tester may not exercise all parts of the code

Testing Methods

- White Box Testing:
  - The tester has access to the underlying implementation and applies tests to satisfy some criteria
    - e.g Code coverage: Writing tests to ensure that all program statements are executed at least once)
- Black Box Testing:
  - The tester has no access to the underlying implementation, but focusses instead upon testing the system to verify that the functional requirements have been met
  - Advantage: Tester is impartial
  - Disadvantage: Tester may not exercise all parts of the code

We will take the perspective of the developer and focus on **white box testing**.

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test
2. Run all tests

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test
2. Run all tests
    1. The new test should fail because we haven't implemented the feature yet!

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test
2. Run all tests
    1. The new test should fail because we haven't implemented the feature yet!
3. Write some code that causes the test to pass

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test
2. Run all tests
    1. The new test should fail because we haven't implemented the feature yet!
3. Write some code that causes the test to pass
4. Run all tests

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test
2. Run all tests
    1. The new test should fail because we haven't implemented the feature yet!
3. Write some code that causes the test to pass
4. Run all tests
5. Refactor code and re-run tests

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test
2. Run all tests
    1. The new test should fail because we haven't implemented the feature yet!
3. Write some code that causes the test to pass
4. Run all tests
5. Refactor code and re-run tests
    1. Clean up the code and apply principles and patterns to remove *code smells* without altering behaviour

Test Driven Development

Agile Software Development methodologies (e.g. Extreme Programming) generally advocate **test driven development (TDD)**. The focus is on unit tests and the basic idea is to write the test for each feature *prior to implementing the feature*. The test driven development cycle is as follows:

1. Add a new test
2. Run all tests
    1. The new test should fail because we haven't implemented the feature yet!
3. Write some code that causes the test to pass
4. Run all tests
5. Refactor code and re-run tests
    1. Clean up the code and apply principles and patterns to remove *code smells* without altering behaviour
6. Repeat

Advantages of TDD

- Encourages more tests to be written, which increases productivity

Advantages of TDD

- Encourages more tests to be written, which increases productivity
- Forces developer to consider the usage of their code by clients

Advantages of TDD

- Encourages more tests to be written, which increases productivity
- Forces developer to consider the usage of their code by clients
- Tests act as **executable documentation** for your code!

Advantages of TDD

- Encourages more tests to be written, which increases productivity
- Forces developer to consider the usage of their code by clients
- Tests act as **executable documentation** for your code!
- Forces developer to decouple components required to run the tests

Advantages of TDD

- Encourages more tests to be written, which increases productivity
- Forces developer to consider the usage of their code by clients
- Tests act as **executable documentation** for your code!
- Forces developer to decouple components required to run the tests
  - "Writing tests before code improves our designs." [Martin(2003)]

Decoupling

In order to write a unit test, we will need to decouple the software unit being tested from other objects.

Decoupling

In order to write a unit test, we will need to decouple the software unit being tested from other objects.

e.g. Test the `payEmployees` method of our Payroll class. Here is our design so far:

Decoupling

In order to write a unit test, we will need to decouple the software unit being tested from other objects.

e.g. Test the `payEmployees` method of our Payroll class. Here is our design so far:

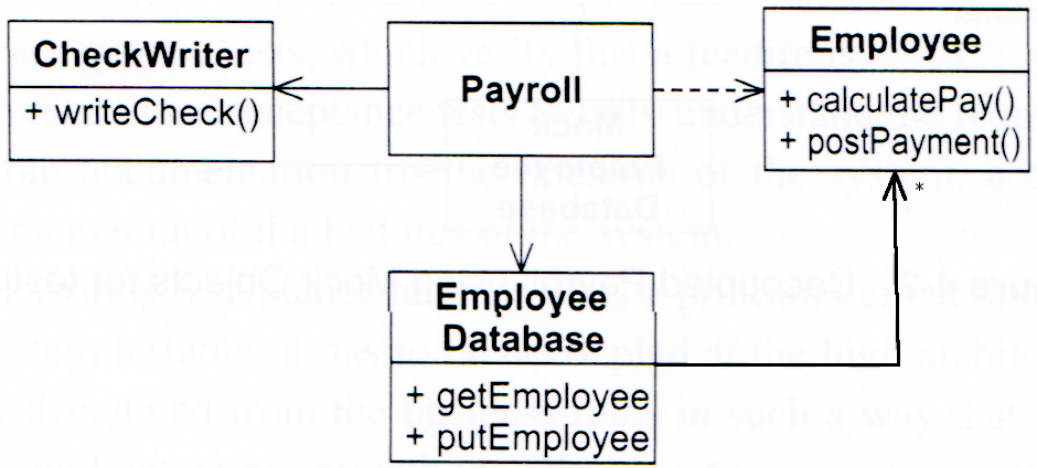

Figure 4-1 Coupled Payroll Model

Decoupling

In order to write a unit test, we will need to decouple the software unit being tested from other objects.

e.g. Test the `payEmployees` method of our Payroll class. Here is our design so far:

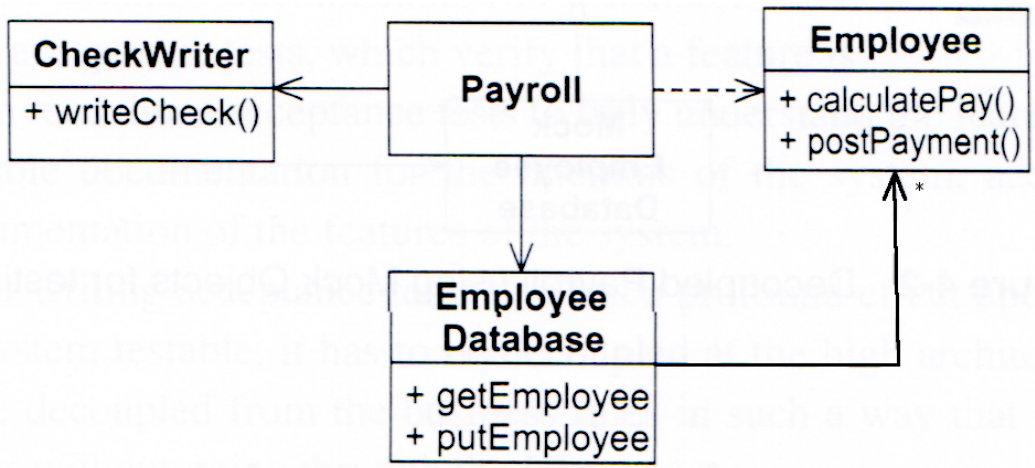

Figure 4-1 Coupled Payroll Model

(Note: modified from [Martin(2003)]).
How can we test Payroll without complete implementations for the other classes? Solution: the Mock Object design pattern.

Create interfaces in place of the other classes and provide mock implementations. Later these mock implementations can be replaced:

Create interfaces in place of the other classes and provide mock implementations. Later these mock implementations can be replaced:

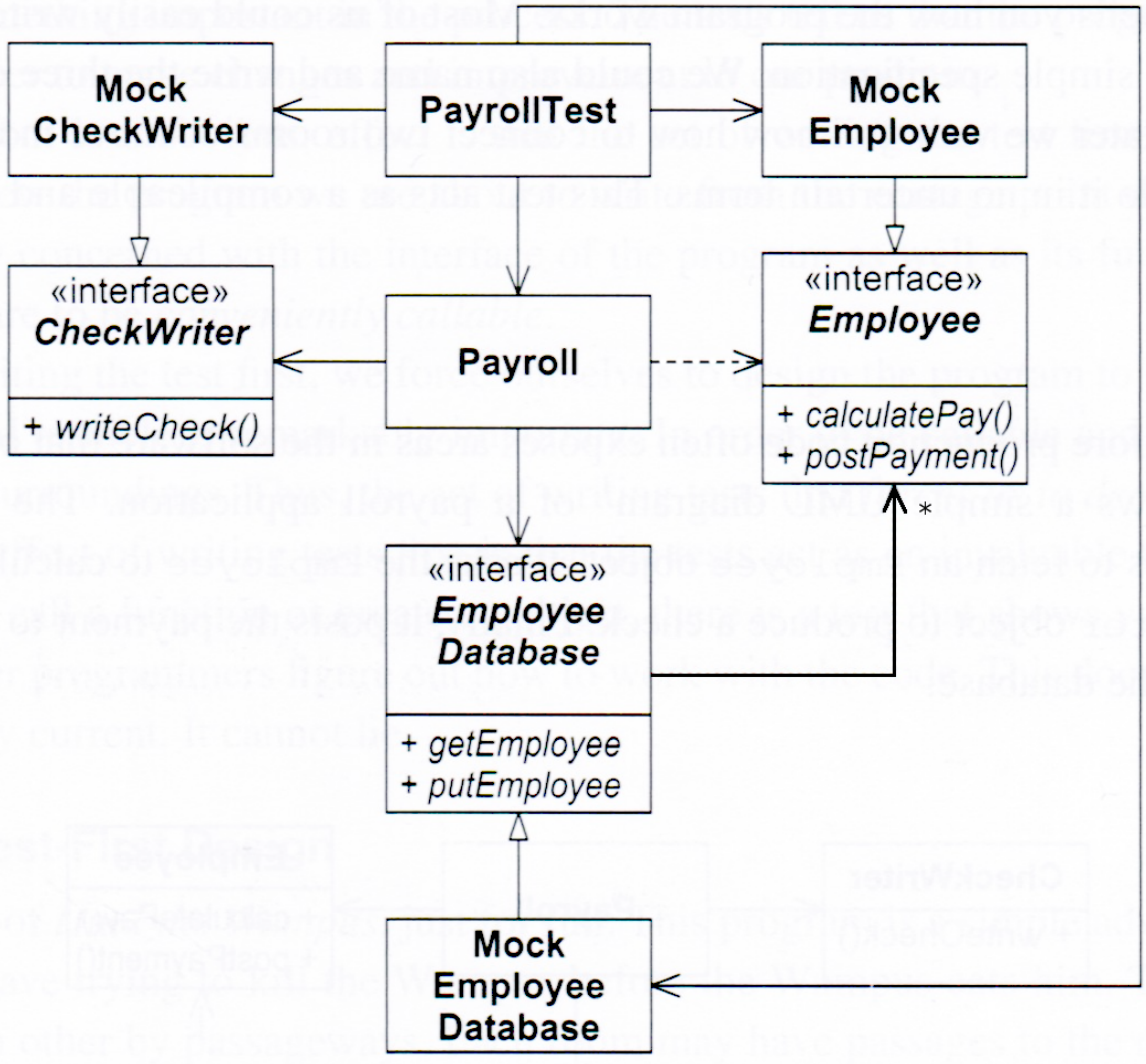

Figure 4-2 Decoupled Payroll using Mock Objects for testing

We can now write our test for payEmployees:

```
public void testPayroll() {
    MockEmployeeDatabase db = new MockEmployeeDatabase();
    MockCheckWriter w = new MockCheckWriter();
    Payroll p = new Payroll(db, w);

    p.payEmployees();

    assert w.checksWereWrittenCorrectly();
    assert db.paymentsWerePostedCorrectly();
}
```

JUnit

JUnit is a unit-testing framework for Java that works nicely with TDD. It is one of a family of testing frameworks known as xUnit (e.g. CPPUnit for C++, PyUnit for Python).

JUnit

JUnit is a unit-testing framework for Java that works nicely with TDD. It is one of a family of testing frameworks known as xUnit (e.g. CPPUnit for C++, PyUnit for Python).
Examples:

- Basic usage:
  - BoundedAngle and TestBoundedAngle
- Using a test fixture:
  - IntVect and TestIntVect

References


Robert C. Martin.
Agile Software Development: Principles, Patterns, and Practices.
Prentice Hall, 2003.