

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

# Program Testing. Refactoring.

Lect. PhD. Arthur Molnar

Babes-Bolyai University

*arthur@cs.ubbcluj.ro*

# Overview

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

## 1 Program testing

- Testing Approaches
- Black-box and White-box Testing
- Testing Levels
- Automated testing
- Debugging

## 2 Refactoring

- Coding style
- Refactoring
- How to refactor

# Program testing

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing  
Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## What is testing?

Testing is observing the behavior of a program over many executions.

# Program testing

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor

- We execute the program for some input data and compare the result we obtain with the known correct result.
- **Questions:**
  - How do we choose input data?
  - How do we know we have run enough tests?
  - How do we know the program worked correctly for a given test? (known as the oracle problem)

# Program testing

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing Approaches  
Black-box and White-box Testing

Testing Levels  
Automated testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor

- Testing cannot prove program correctness, and cannot identify all defects in software. However, what it **can** prove is incorrectness, if at least one test case gives wrong results.
- **Problems with testing**
  - We cannot cover a function's input space
  - We have to design an oracle as complex as the program under test
  - Certain things are practically outside of our control (e.g. platform, operating system and library versions, possible hardware faults)

# Testing Approaches

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches

Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor

## Exhaustive testing

- Check the program for all possible inputs.
- Impractical for all but mostly trivial functions.
- Sometimes used with more advanced techniques (e.g. symbolic execution) for testing small, but crucial sections of a program (e.g. an operating system's network stack)

# Testing Approaches

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches

Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

## Boundary value testing

- Test cases use the extremes of the domain of input values, typical values, extremes (inside and outside the domain).
- The idea is that most functions work the same way for most possible inputs, and to find most of those possibilities where functions use different code paths.

# Testing Approaches

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches

Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing

Debugging

Refactoring

Coding style

Refactoring

How to refactor

## Random testing, pairwise (combinatorial) testing, equivalence partitioning

- And the list goes on...



# Testing Methods

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor

## Black box testing

- The source code is not available (it is in a "black", non-transparent box)
- The selection of test case data for testing is decided by analyzing the specification.

## White box testing

- The source code is readily available (it is in a transparent box) and can be consulted when writing test cases.
- Selecting test case data is done by analyzing program source code. We select test data such that all code, or all execution paths are covered.
- When we say "*have 80% code coverage*" (Lab5-7, bonus) it is white-box testing.

# Demo

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches

Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## White and Black-box testing

Examine the test code in **20-BlackBoxWhiteBox.py**

# Advantages and drawbacks

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

## Black box testing

- + Efficient for large code-bases
- + Access to source code is not required
- + Separation between the programmer's and the tester's viewpoint
- You do not know how the code was written, so test coverage might be low, testing might be inefficient

# Advantages and drawbacks

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches

Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## White box testing

- + Knowing about the code makes writing it **AND** testing it easier
- + Can help find hidden defects or to optimize code
- + Easier to obtain high coverage
  - Problems with code that is completely missing
  - Requires good knowledge of source code
  - Requires access to source code

# White and Black-box testing

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches

**Black-box and  
White-box  
Testing**

Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

**NB!**

It's not a matter of which box is better, it's more like you have to make do with what you've got!

# Testing levels

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

**Testing Levels**  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## Testing Levels

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test

# Testing levels

## Lecture 10

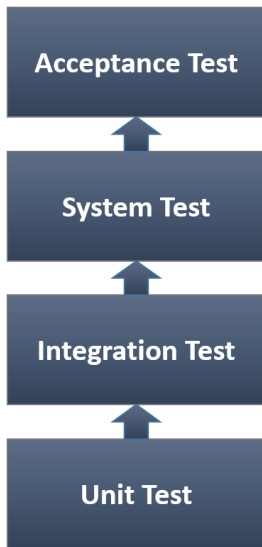
Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor



## Unit Test

- Refers to tests that verify the functionality of a specific section of code, usually at function level.
- Testing is done in isolation. Test small parts of the program independently

## Integration Test

- Test different parts of the system in combination
- In a bottom-up approach, it is based on the results of unit testing.

# Testing levels

## Lecture 10

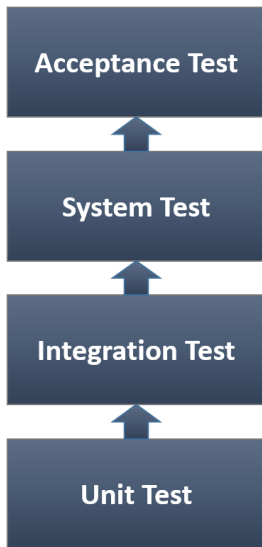
Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor



## System Test

- Considers the way the program works as a whole.
- After all modules have been tested and corrected we need to verify the overall behavior of the program

## Acceptance Test

- Check that the system complies with user requirements and is ready for use



# Testing levels

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

**Testing Levels**  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor

## Remember!

- What you did in Labs 2,3,5 is unit testing.
- When you checked that your program worked through its UI, it was integration/system testing.
- What we did during Lab 4 was acceptance testing.

# Automated testing

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
**Automated  
testing**

Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## Automated testing

- Test automation is the process of writing a computer program to do testing that would otherwise need to be done manually.
- Use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions

# PyUnit - Python unit testing framework

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
**Automated  
testing**

Debugging


### Refactoring

Coding style  
Refactoring  
How to refactor

## The unittest<sup>1</sup> module supports:

- Test automation
- Sharing of setup and shutdown code for tests
- Aggregation of tests into collections
- Independence of tests from the reporting framework

---

<sup>1</sup><https://docs.python.org/3/library/unittest.html> 

# Demo

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
**Automated  
testing**  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## PyUnit

Run the test code in the following file in Eclipse, using a  
*Python unittest* run; **21-PyUnitTest.py**

# PyUnit - Python unit testing framework

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor

## The unittest module supports:

- Tests are implemented using classes derived from **unittest.TestCase**
- Test methods should start with the characters **test**
- We now use special methods instead of **assert** statements directly - **assertTrue()**, **assertEqual()**, **assertRaises()** and many more<sup>2</sup>.
- The **setUp()** and **tearDown()** methods are run before and after each test method, respectively.

---

<sup>2</sup><https://docs.python.org/3/library/unittest.html#assert-methods>

# Automated testing

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing  
Testing Levels  
**Automated  
testing**  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## Discussion

How can we know when our test are "good enough" ?

# The Coverage module

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
**Automated  
testing**

Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## One (of the simpler) ways is to use code coverage

- Measure how much of the entire code was executed during the tests
- 0% coverage means no lines of code were executed
- 100% means **ALL** lines of code were executed at least once
- There exist tools which can measure and report this automatically

# Installing code coverage for Eclipse

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
Refactoring  
How to refactor

## The coverage module<sup>3</sup> gathers coverage information that can be displayed in Eclipse

- 1 Install the coverage module using *"pip install coverage==3.x"* (x depends on your Eclipse and PyDev versions, try 3.3, 3.6 or 3.7)
- 2 Refresh your Python interpreter info in Eclipse to see the new module
- 3 Make sure the source code is in a directory with no spaces
- 4 In the *Code Coverage* view in Eclipse, enable the *"Enable code coverage for new launches"* checkbox
- 5 ?
- 6 Now it should work!

---

<sup>3</sup><https://coverage.readthedocs.io/en/coverage-4.2/>



# Demo

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
**Automated  
testing**  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## PyUnit with coverage

After installing the coverage module, try to gather coverage for the test cases in example **21-PyUnitTest.py**

# Debugging

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing  
Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

## Debugging

The activity that must be performed when testing indicates the presence of errors, to identify errors, and rewrite the program with the purpose of eliminating them.

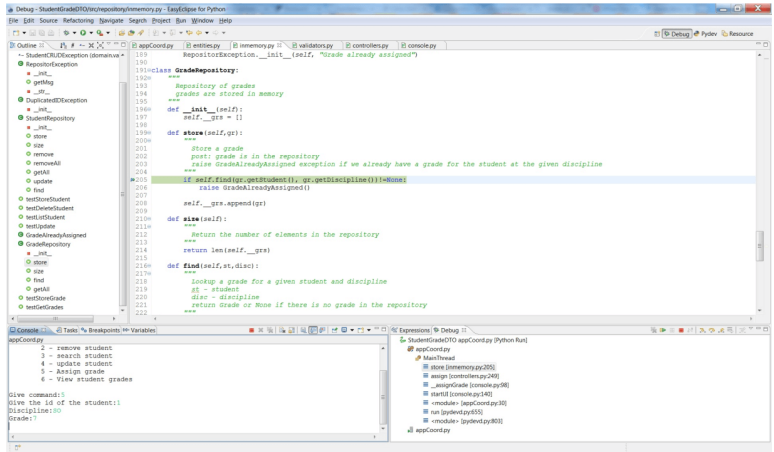
### Two major approaches to debugging

- Using print statements
- Using the IDE

## Eclipse debug perspective - Example

## Lecture 10

Lect. PhD.  
Arthur Molnar



# Eclipse debug perspective

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

## Debug view

- View the current execution trace (stack trace)
- Execute step by step, resume/pause execution

## Variables view

- View variable values

# Program inspection

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

- Anyone can write code that computers understand. It's about writing code that humans also understand!
- Programming style consist of all the activities made by a programmer for producing code easy to read, easy to understand, and the way in which these qualities are achieved

# Program inspection

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

- Readability is considered the main attribute of style.
- A program, like any publication, is a text must be read and understood by another programmer. The element of coding style are:
  - Comments
  - Text formatting (indentation, white spaces)
  - Specification
  - Good names for entities (classes, functions, variables) of the program
    - Meaningful names
    - Use naming conventions

# Naming conventions

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

- Class names: Student, StudentRepository
- Variable names: student, nrElem (nr\_elem)
- Function names: getName, getAddress, storeStudent  
(get\_name, get\_address, store\_student)
- constants: MAX

Whatever convention you use, use it **consistently**.

# Refactoring

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

## Refactoring

The process of changing the software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.

- It is a disciplined way to clean up code that minimizes the chances of introducing bugs.
- When you need to add a new feature to the program, and the program's code is not structured in a convenient way for adding the new feature, first refactor the code to make it easy to add a feature, then add the feature



# Why refactoring

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing  
Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
**Refactoring**  
How to refactor

- Refactoring improves the design of the software
- Refactoring makes software easier to understand
- Refactoring helps you find bugs
- Refactoring helps you program faster

# Bad smells

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring  
Coding style  
**Refactoring**  
How to refactor

## When is refactoring needed?

- Duplicated code
- Long method/class
- Long parameter list (more than 3 parameters is seen as unacceptable)
- Comments

# Refactoring methods

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

- **Rename Method** - *The name of a method does not reveal its purpose.*
- **Consolidate Conditional Expression** - *You have a sequence of conditional tests with the same result.*  
Combine them into a single conditional expression and extract it.
- **Consolidate Duplicate Conditional Fragments** - *The same fragment of code is in all branches of a conditional expression.* Move it outside the expression.

# Refactoring methods

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

- **Decompose Conditional** - *You have a complicated conditional (if-then-else) statement.* Extract methods from the condition, then part, and else parts.
- **Inline Temp** - *You have a temp that is assigned to once with a simple expression, and the temp is getting in the way of other refactorings.* Replace all references to that temp with the expression.
- **Introduce Explaining Variable** - *You have a complicated expression.* Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.

# Refactoring methods

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing  
Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

- **Remove Assignments to Parameters** - *The code assigns to a parameter. Use a temporary variable instead.*
- **Remove Control Flag** - *You have a variable that is acting as a control flag for a series of boolean expressions. Use a break or return instead.*
- **Remove Double Negative** - *You have a double negative conditional. Make it a single positive conditional*

# Refactoring methods

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

- **Replace Nested Conditional with Guard Clauses** - *A method has conditional behavior that does not make clear what the normal path of execution is. Use Guard Clauses for all the special cases.*
- **Replace Temp with Query** - *You are using a temporary variable to hold the result of an expression. Extract the expression into a method. Replace all references to the temp with the expression. The new method can then be used in other methods.*

# Refactoring classes

## Lecture 10

Lect. PhD.  
Arthur Molnar

Program  
testing

Testing  
Approaches  
Black-box and  
White-box  
Testing

Testing Levels  
Automated  
testing  
Debugging

Refactoring

Coding style  
Refactoring  
How to refactor

- **Encapsulate Field** - *There is a public field.* Make it private and provide accessors.
- **Replace Magic Number with Symbolic Constant** - *You have a literal number with a particular meaning.* Create a constant, name it after the meaning, and replace the number with it.
- **Extract Method** - *You have a code fragment that can be grouped together.* Turn the fragment into a method whose name explains the purpose of the method.

# Refactoring classes

## Lecture 10

Lect. PhD.  
Arthur Molnar

### Program testing

Testing  
Approaches  
Black-box and  
White-box  
Testing  
Testing Levels  
Automated  
testing  
Debugging

### Refactoring

Coding style  
Refactoring  
How to refactor

- **Move Method** - *A method is, or will be, using or used by more features of another class than the class on which it is defined.* Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.
- **Move Field** - *A field is, or will be, used by another class more than the class on which it is defined.* Create a new field in the target class, and change all its users.