Implementation

(the fun bit)

Dr. Julie Greensmith

Overview

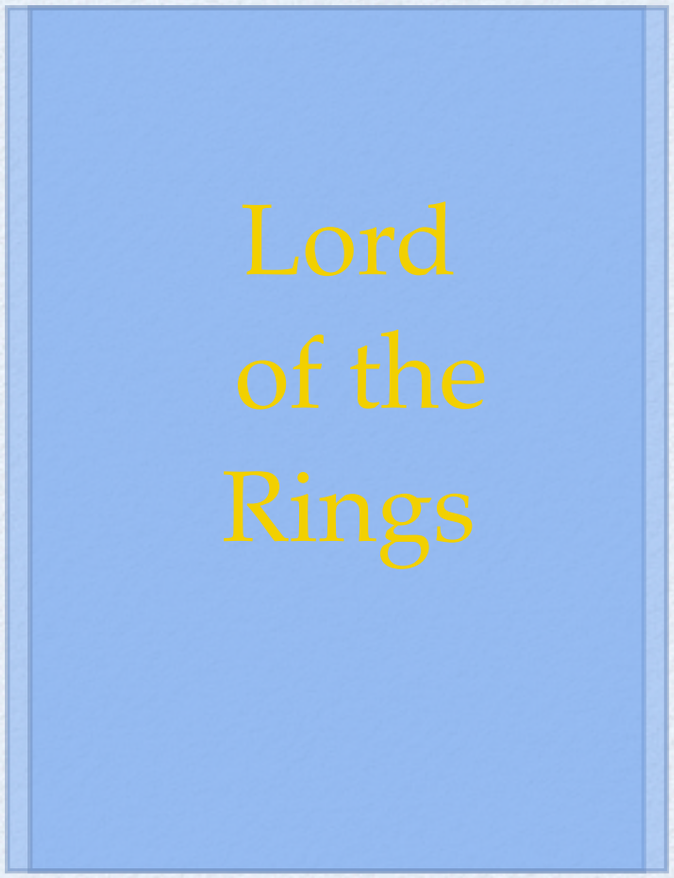
- Object Orientation Recap
- Creating a top-down view
- Things to consider for a good implementation
- Development environments and tools
- A Rapid Prototyping approach

Revising Objects

- Objects/Classes consist of three parts:
  - Identity - names
  - Data - variables
  - Code - functions/methods
  - Access only through class message interface
- *How do we derive objects from a problem description?*

Objects...

- Identity is determined by identifying the **nouns**
  - e.g. book object
- Behaviours are defined by identifying the **verbs**
  - e.g. buy, sell, search
- Class Book has instance “50 Shades”, we can sell the book



Class: Book

Instance: Lord of the Rings

State: unsold

Behaviour: add to cart

What do we have so far?

- We understand our user and what a system is for
  - identified the stakeholders
- Gathering and formalisation of requirements through requirements engineering
- Development of a Software Requirements Specification
- Design of what you want to implement through some type of model

The implementation step

- From our design we then actually want to make something!
  - we should have already done some form of design
  - some form of model (perhaps UML or a formal model)
  - could be in our heads or on paper
- What's the next step?
  - realising our design with an implementation
  - number of issues involved in the implementation step

The usual approach

- Develop top down
  - start with your classes or units you wish to develop
  - know which data structures you are going to use
  - some UML development programs can generate this skeleton code for you e.g. Rational Rose
- Once you have the macro structure (or skeleton) implemented then start filling in the details
  - types, return values
  - Finally add your implementation code e.g. if statements and conditions

For a complete design?

- No problem, simply verify your code does what you expect through a series of *unit tests*
- However, it is rare for the design to be complete and/or totally correct the first time through
  - rework the design as you become more aware of the technical challenges of the implementation
  - modify your code as the specification has changed
  - dependent on time and cost constraints

How to accommodate for change

- Write code that is easy to maintain and modify
  - not just by yourself but for other people too
- A number of factors that can help
  - develop code in a consistent style
  - conform to an agreed coding standard
  - add appropriate comments

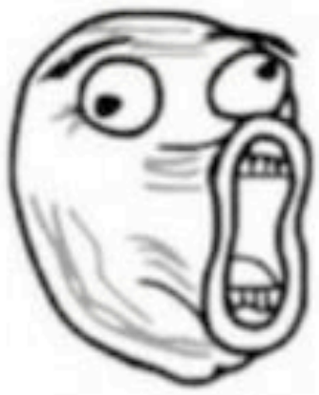
```
import random
import unittest

def shuffle(cards):
    max = len(cards)-1
    while max != 0:
        r = random.randint(0, max)
        cards[r], cards[max] = cards[max], cards[r]
        max = max - 1
    return cards

class TestCase(unittest.TestCase):
    def setUp(self):
        self.actual = range(1, 53)

    def test_elements(self):
        expected = shuffle(self.actual)
        self.assertEqual(set(self.actual),
set(expected))

if __name__ == "__main__":
    unittest.main()
```

Consistency and coding standards

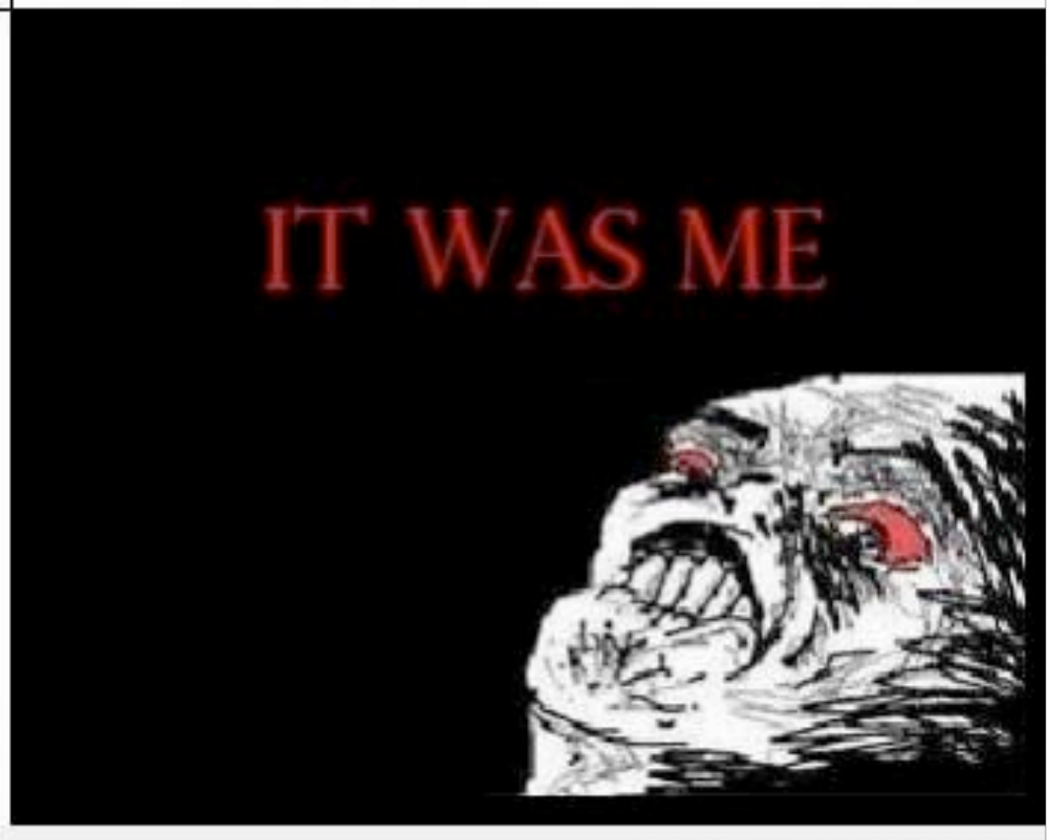
- Most companies or development teams have set coding standards and conventions for how a program should be developed - *in house style*
- Eg. variable name formats
  - a_variable_name, my_function
  - AVariableName, MyFunction
  - avariablename, myfunction
- Hungarian notation is one agreed standard
- It doesn't matter which standard you use - it will still compile!
- Consistent code is much easier to work with, read and modify in the future

Opening source code

Hmmmm... this hardly looks like it should work



LOL! Who wrote this, It's horrible!! Let me check...



memecenter.com

Comments

- It sounds so simple, add comments to your code to make it more readable
  - you wont remember what that function does in 6 months time!
- The ‘lazy coder’ will often think that comments are not needed and just want to get on with writing the code itself
  - does not benefit the lazy coder as they will have to work twice as hard the next time they look at the code
- Roughly, if its not completely obvious what the code does, it needs some sort of comment
- I start off with comments to get my structure in place!

Be consistent with coding tools

- Know which type and version of compiler you need to use
  - different versions can behave in different ways, require different libraries
- Same goes for external libraries
  - might want to write it in the comments of the code!
- For larger problems might want to develop a testing rig

Assistance in developing code

Terminal — bash — 80×24

```
#include <stdio.h>

int main()
{

        float num1, num2;
        char operation;
        scanf("%f%c%f", &num1, &operation, &num2);

        if(operation == '+')
                printf("%f\n", num1+num2);

        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
"basiccalc.c" 14 lines, 180 characters
```

For small quick programs

- If you have only one file and want to write something quickly easiest thing to do is to use a tool like GVim or Emacs
- Compile on the command line perhaps using a tool such as "Make"
- Very powerful little editors which let you do things like syntax highlighting and auto-indentation
  - gets a bit confusing if you have tons of files and zillions of variable names

Use an IDE to help

- IDE = Integrated Development Environment
- Assist in giving you multi-level overviews of your code, automatic completion of variable names, and have the compiler built in to help speed up the development process
- Widely used as the accepted tool of commercial software development
- MS Visual Studio probably the most popular and frequently used
- But there are tons of these available
  - Eclipse, Xcode (mac), netbeans...

Checking for errors

- Don't be precious about your code, accept that you will make mistakes no matter how careful
- Checking for common errors while you are coding really helps
- Memory leaks are notorious for causing problems
  - some compilers (like the C++ compiler in Visual Studio 2003) do not check for this and so you end up running out of memory!
  - Can use a checker for your code called *Lint* which will check for common sources of memory leak


MEMORY LEAKS
MEMORY LEAKS EVERYWHERE
memegenerator.net

Debugging

- Many commercial and open source debugging tools exist - use them!
- Can watch variables, make sure they are the correct value
- Can set breakpoints to make sure you are actually getting into the correct parts of the program
- Can check that you have not made hideous logic errors
- Common ones include gdb and ddd, but some IDE's have them built-in.

So we have written our code...

- Check back against the specification to make sure you have actually implemented what you were supposed to implement!
  - contact your requirements engineer to make sure the client has not added any extra functions!
- Go through a rigourous process of testing
- Perform validation and verification of the code

Important things to check

- Performance:
  - is your code efficient?
  - does it run at a fast enough speed?
- Optimisation:
  - some prefer to optimise their code while they write
  - big mistake!
  - “Premature optimisation of code is the root of all evil, well not quite, but it is certainly the source of the majority of bugs”

What to do if its not good enough

- Go through a process called "refactoring"
  - update and clean up the non-functional elements of your code
  - increases code stability and often readability
- Once you have the functionality working, don't be afraid to throw your code away if you can now think of a better way of achieving the same!
  - this is the process of prototyping

Dont overcook!

- Use prototyping when implementing your code
  - it will not be right first time
  - dont be afraid to throw it away once you have the general idea
- Programs, especially large ones, are in flux when you first construct them
- Keep it small, keep it simple, keep checking for errors