

Object-Oriented Analysis and Design Course Book

T++ Technical Skills Training Program

CUNY Institute for Software Design & Development (CISDD)

New York Software Industry Association (NYSIA)

Aron Trauring

Class 1 April 11th, 2005

Class Agenda

9:00-10:30 Class Introductions / What this Course Is About (Presentation)

10:30-11:15 Exercise 1 — Building a Simple Model

11:15-11:45 Break Out Presentations

11:45-1:00 Process Basics (Presentation)

1:00-2:00 Lunch

2:00-3:15 Core Object-Oriented Concepts — Part I (Presentation)

3:15-4:15 Core Object-Oriented Concepts — Part I (Presentation)

4-15-5:00 Core OO Concepts from a Programming Perspective — Python — Part I

Instructor

Aron Trauring – CEO, Zoteca

Email: atrauring@zoteca.com

Personal Website: <http://aronst.org/>

Zoteca Corporate Website: <http://www.zoteca.com/>

FOSS Resources: <http://www.fourm.info/>

What This Course Is

What This Course Is Not

- UML as Documentation
- UML as Programming Language
- Certification Prep
- Introduction to Rational Rose
- UML tools how to

How We Develop Software Matters

- Saves Money
- Save Jobs
- Save Lives

Course Biases

- Agile Development
- How to think and design in objects
- UML as sketching

Course Biases — Agile Development

- Agile Process
- Minimal Artifacts

Course Biases — How to think and design in objects

- GRASP
- Patterns

Course Biases — UML as sketching

- Bare minimum syntax
- Bare minimum charts
- Other tools

IBM Certification

- IBM Object-Oriented Analysis and Design with UML <http://www-03.ibm.com/certify/tests/edu486.shtml>

Course Outline

I. Process Basics

- Agile
- UP
- XP

II. Core Principles of OOAD

III. Analysis Techniques

- Use Cases
- Domain Models
- System Sequence Diagrams

III. High Level Design Techniques

- Logical Architecture (Package Diagrams)
- GRASP
- Class Diagrams
- Interaction Diagrams
- GOF Patterns

Bibliography

Required Text:

UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition by Martin Fowler

Core Curriculum Texts:

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd Edition by Craig Larman

Design Patterns Explained: A New Perspective on Object-Oriented Design (Software Pattern Series), Second Edition by Alan Shalloway and James Trott

Writing Effective Use Cases by Alistair Cockburn

Practical Python by Mark Lutz

Other Useful Books:

Agile and Iterative Development: A Manager's Guide by Craig Larman

The Rational Unified Process: An Introduction by Philippe Kruchten

Extreme Programming Explained: Embrace Change by Kent Beck

Online Bibliography: <http://www.fourm.info/TPP/OOAD/bibliography.html>

Exercise 1 - Building a Simple Model

1. We are familiar with the game “Go Fish.” Here are the rules:

This game, often just known as Fish, is best for 3-6 players, but it is possible for 2 to play. A standard 52 card deck is used. The dealer deals 5 cards to each player (7 each for 2 players). The remaining cards are placed face down to form a stock.

The player to dealer’s left starts. A turn consists of asking a specific player for a specific rank. For example, if it is my turn I might say: ‘Mary, please give me your jacks’. The player who asks must already hold at least one card of the requested rank, so I must hold at least one jack to say this. If the player who was asked (Mary) has cards of the named rank (jacks in this case), she must give all her cards of this rank to the player who asked for them. That player then gets another turn and may again ask any player for any rank already held by the asker. If the person asked does not have any cards of the named rank, they say ‘Go fish!’. The asker must then draw the top card of the undealt stock. If the drawn card is the rank asked for, the asker shows it and gets another turn. If the drawn card is not the rank asked for, the asker keeps it, but the turn now passes to the player who said ‘Go fish!’.

As soon as a player collects a book of 4 cards of the same rank, this must be shown and discarded face down. The game continues until either someone has no cards left in their hand or the stock runs out. The winner is the player who then has the most books.

Create a simple descriptive model for this game. Work in groups of 4-5 people.

2. Analyze what techniques you are using to model, i.e. meta-model.

3.. One person from group will present to the whole class.

Process Basics

Early Days

- The Fifties - Hardware dominates
- High level languages - Fortran, Cobol
- Small and efficient programs
- Compiler tricks

The Sixties - The rise of software

- 1965 G. Moore's (Intel) Law - 'The density of chips doubles every year'
- 1968 Data General Nova - 32K \$8,000
- Software costs exceed hardware
- OS/360 + High level languages - programs portable and durable

New Criteria for Successful Software Development

- A relatively low cost of initial development.
- Easily maintained.
- Portable to new hardware.
- Does the job the customer wants.

Structured Programming, Analysis and Design

- Developing programs top-down (as opposed to bottom-up).
- Using a set of specific formal programming constructs (the go to was to be banished).
- Following some formal steps to decompose the larger problem.
- Human readable vs. Machine readable
- Software Engineering
- Methods Wars
- Guru Consultants - Edward Yourdon, Peter Coad, James Martin, and Tom DeMarco

Fred Brooks - The Mythical Man Month

- PM of IBM's Operating System/360 (OS/360) in the early 1960's.
- Software development a human-centric process, not an engineering discipline.
- Why programming is hard to manage
- Why Projects Fail

The Mythical Man Month Method (FourM)

- Reduce Complexity
- Enhance Communication
- Project Design - Conceptual Integrity
- Project Structure - The Surgical Team
- Project Implementation - Iterative and Incremental Development (IID)
- Project Communications — The Documentary Hypothesis
- Project Organization — Plan the Organization for Change

SDLC — Project Phases

1. Requirements — What the Customer wants
2. Analysis — Understanding the Requirements — Functional Specification
3. Design — Software Architecture/Decomposition — Technical Specification
4. Development/Implementation — Writing the code — Programs
5. Unit Test — working unit
6. System Test — working system
7. Acceptance Test — Working application

Eighties — New Directions

- SDLC a failure
- Projects grow more complex
- Search for new paradigms

OOA/D — Key Concepts

- Objects basic building blocks
- Objects interact through exchanging messages
- Internal behavior is hidden
- Divide and Conquer
- Close connection between domain model and implementation

OOA/D — UML

- Structured Analysis gurus picked up ideas of OO especially with DOD backing of ADA in 1983
- Notational and methods wars raged for fifteen years
- 1997 Object Management Group issued first UML standard
- Standardized notation for modeling software systems
- A tool for improving communications
- UML is a notation not a method

OOA/D — UP

- Three amigos — three gurus, Grady Booch, Ivar Jacobson, and Jim Rumbaugh
- Rational Systems — now owned by IBM
- RUP or UDP
- High ceremony tradition but influenced by agilists

UP — Key Practices

- Develop in short time-boxed iterations
- Develop high-risk, high-value elements first
- Focus on customer value — Milestone goals
- Accommodate change early

The Agile Manifesto — February, 2001

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Iterative Development

- Overall life-cycle is composed of multiple iterations
- Each iteration goes through all waterfall activities
- Iteration release at end of each cycle - stable, tested, integrated, partially complete

Iterative Incremental Development

- Each iteration adds new features
- Number of iterations varies with project and method
- Not the same as prototyping
- IID is 100% FourM compatible
- All Agile methods are IID
- Agile <> IID
- Older IID methods (e.g. RUP) are adapting to/adopting Agile concepts

XP — Historical Background

- Kent Beck and Ward Cunningham (Tektronix research)
- CRC cards
- Design Patterns (Smalltalk)
- Cunningham invented the Wiki
- C3 payroll project at Chrysler (Martin Fowler, Ron Jeffries)

XP — Key Practices

- On-site customer as part of team
- Extreme avoidance of up-front design work
- Automated acceptance tests / test-driven development
- Pair programming

Core Object-Oriented Concepts

The Fundamental Challenge of Software Development

- Change is the only constant
- How do we deal with change?

It Starts With Requirements

- Requirements are incomplete
- Requirements are usually wrong
- Requirements (and users) are misleading
- Requirements do not tell the whole story

Why Requirements Change

- The users' view of their needs change as a result of their discussions with developers and from seeing new possibilities for the software
- The developers' view of the users' problem domain changes as they develop software to automate it and thus become more familiar with it
- The environment in which the software is being developed changes

Authoritarian Model

- Change is a necessary evil
- Manage change by controlling it
- Top-down approaches - Structured Analysis/Design/Programming
- SDLC, UP
- Ada School of OO — OO is about Information Hiding

Anarchist Model

- Embrace change
- Allow maximum flexibility to both encourage and manage change
- Bottom-up approaches
- Agilists, XP
- Smalltalk school of OO — OO is about “Those Who Know, Decide”

Functional Decomposition

- Break down (decompose) problem into functional steps
- Repeat and rinse
- Recipe model

Mini-Exercise 1

“Access a set of shapes stored in a database and then display them”

Problem with Structured Programming

- One main controller program
- “With great power comes great responsibility”
- Too much responsibility — code gets complicated
- Lesson 1: need to delegate responsibility
- All changes impact main module
- Focusing on too many things in one place means more error prone
- Lesson 2: Keep things simple
- Modularity can't deal with variable and changing data structure for shapes
- Lesson 3: Those who know should deal

Cohesion and Coupling

- Cohesion — how strongly the internal contents of a routine are related to each other
- Coupling — how strongly a routine is related to other routines
- Functional decomposition leads to weak cohesion and strong coupling
- Unwanted Side Effects — change in one place effects another unknown place
- Most of the time in debugging is spent on finding unwanted side effects or understanding how to avoid them
- Goal: strong cohesion (internal integrity) and loose coupling (small, direct and visible relations)

Mini exercise 2

“You are an instructor at a conference. People in your class had another class to attend following yours, but didn’t know where it was located. One of your responsibilities is to make sure everyone knows how to get to their next class.”

Difference Between SP and OO

- SP — giving directions to everyone — you have to pay close attention to too many details. No one but you is responsible. You go crazy!
- OO – you give general instructions and then assume each person will figure out how to the task individually.
- Shift responsibility to those who know.

Mini exercise 3

“Need to give special instructions to graduate students who are assisting at the conference. Perhaps they need to collect course evaluations and take them to the conference office before they can go to the next class.”

Accommodating Change

- SP: every type of new student requires a change to controller program
- OO: new categories of students don’t effect the main program; students them-self responsible

Three Perspectives on Software Systems

- Conceptual — “What am I responsible for?”
- Protocol (Interface) — “How am I used?”
- Implementation — “How do I fulfill my responsibilities?”

Three Sources of Flexibility

1. Conceptual — Students responsible for own behavior
2. Protocol (Interface) — Control program can talk to different students as if they were the same
3. Implementation — Control program doesn’t need to be aware of the internal behaviors that lead to desired outcome

Abstraction — A Human-Centric Need

- In the beginning Binary machine code: 100101010100.....
- Hex: 0A DE 01.....
- Assembler:

```
MOV DPTR, #CNT
```

- Third generation:

```
Do
Portc = Inp(pdc)  ' Reads port C
Out Pda , Portc  ' Writes it to port A
Loop
```

- OOPL:

```
SJ = RegularStudent('Joe')
SJ.GoToNextClass()
```

What is an Object?

- Conceptual — set of responsibilities — **Student**
- Protocol (Interface) — set of operations (behaviors) that can be invoked by other objects or itself — **GoToNextClass**
- Implementation — the specific data structures and related methods that implement the behaviors — **RegularStudent.GoToNextClass**

Classes and Instances

- Abstract Class — Highest level of an object — **Student**
- Concrete Class — *Subclasses* — more specific — **RegularStudent, GraduateStudent**
- Instance — Concrete items in a programs domain — *Instantiation* of a Class — **SJ = RegularStudent('Joe')**

Abstraction Benefit — Model Objects at Three Levels

- Analysis — Requirements of System under Design — Domain Model
- Design — Structure of software system — Domain Layer
- Implementation — Working Code — Language specific
- Goal: Small gap as possible between all three levels

Inheritance Benefit of Objects

- Set of attributes passed to descendants
- Conceptual — Is-A Relationship — Defines sets of responsibilities down the tree — **GoToNextClass**
- Protocol (Interface) — Customization / Specialization — Sub-classes can replace, provide or extend behaviors of parent classes — **RegularStudent.GoToNextClass, GraduateStudent.GoToNextClass**
- Implementation — Code reuse

Polymorphism Benefit of Objects

- “Many forms”
- Conceptual — Delegation of responsibility down the tree
- Protocol (Interface) — “many forms” of methods for same operation (depends on instance) — **RegularStudent.GoToNextClass, GraduateStudent.GoToNextClass**
- Implementation — overloading operators or operations — “+”, “print”, “__init__”

Encapsulation Benefits

- Hiding
- Conceptual — Composition — Has-a-Relationship — Classes
- Protocol (Interface) — hiding implementation details — **def GoToNextClass:**
- Implementation — properties hide accessor attributes — **Grade = property(getGrade, setGrade)**

Core OO Concepts from a Programming Perspective — Python — Part I

What is Python?

- Educational Language — Guido von Rossum — Early '90s

"Python aims to encourage the creation of reusable code. Even if we all wrote perfect documentation all of the time, code can hardly be considered reusable if it's not readable."
— Guido von Rossum

- Object Oriented — Extremely Anarchistic
- “Agile Programming Language”
- Interpreted
- Byte-code compilation (similar to Java)

Python Advantage — Quicker to Develop

- 5-10 Shorter than equivalent C++
- Interpreted — eliminates compile, debug cycle
- Large savings in programmer costs

Python Advantage — Easier to Maintain

- Readability
- Large savings in programmer costs

Python Advantage — Versatile

- From scripting to mission critical
- Large savings in training and tools costs

Python Advantage — Platform Portability

- All major OSes and lots of minor ones too
- From PDAs to mainframes
- Write once deploy anywhere

Python Advantage — Language Compatibility

- Microsoft C# and Java camps IronPython (.NET CLR) and Jython
- C, C++ extensions and embedding (original CPython)
- Python as scripting wrapper

Python Advantage — Standards Compatibility

- Internet Standards
- CORBA, COM, SOAP and XML extensions

Python Advantage — Platform Compatibility

- Option for low-level API integration
- Java — lowest common denominator
- C# — Microsoft only

Python Advantage — FOSS

- Real settings for real applications
- Free speech, beer and markets

Python Advantage — Powerful and Easy OO Paradigm

- Dynamic typing and binding
- Everything is an object
- Object behavior can be adapted at run-time
- “Pure” OO Model
- Power of Smalltalk but much easier to learn and use

Python — Myths

- Few Python programmers — thousands and growing
- Interpreted is inefficient — C for time-sensitive
- Just a scripting language — full platform
- No support — commercial tools, training and consultancies
- No one using it — NASA, NATO, Google, IBM, Disney
- No important frameworks — Zope, Plone, Twisted

Why Python for this Course?

- This is *not* an OO programming class
- Need text language for OO modeling
- Need language to illustrate OO concepts
- Python is by far easiest to learn (which is why I know it!)
- Python is by far easiest to read (don't have to be expert to understand)

- Python purest form of OO so best for illustrating concepts
- No common language for students — Python only one compatible with all others
- Learning a bit of Python will give you additional marketable skills

Python in the News

- Python in the Enterprise *<http://programming.newsforge.com/article.pl?sid=05/03/29/0747230>

Mini Exercise 1

“Print Hello World” in the language you know best