OOP

The Easy Way

Graham Lee

OOP the Easy Way

Graham Lee

This book is for sale at http://leanpub.com/ooptheeasyway

This version was published on 2018-10-30

Contents

Introduction

What is object oriented programming? My guess is that object oriented programming will be in the 1980's what structured programming was in the 1970's. Everyone will be in favor of it. Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programmer will practise it (differently). And no one will know just what it is.

Tim Rentsch, Object oriented programming[1]

Object-Oriented Programming has its beginnings in the simulation-focussed features of the Simula programming language, but was famously developed and evangelised by the Smalltalk team at Xerox's Palo Alto Research Center. They designed a computing system intended to be personal, with a programming environment accessible to children who could learn about the world and about the computer simultaneously by modelling real-world problems on their computer.

I recently researched the propagation and extension of Object-Oriented Programming from PARC to the wider software engineering community, which formed the background to my dissertation "We Need to (Small)Talk: object-oriented programming with graphical code browsers"[2]. What I found confused me: how had this simple design language for children to construct computer programs become so complicated and troublesome that professional software engineers struggled to understand it before declaring it a failure and reaching for other paradigms?

[1]https://dl.acm.org/citation.cfm?id=947961

[2]https://www.academia.edu/34882629/We_need_to_Small_talk_object-oriented_programming_with_graphical_code_browsers

A textbook on my shelf, “A Touch of Class” by Bertrand Meyer, claims to be “a revolutionary introductory programming textbook that makes learning programming fun and rewarding”. At 876 pages, it makes it a good workout too: not for the schoolchild, but for the “entering computer science student” at degree level.

Digging further showed that the field of Object Thinking, Object Technology, Object-Oriented Programming or whatever you would like to call it had been subject to two forces:

1. Additive complexity. Consultants, academics and architects keen to make their mark on the world had extended basic underlying ideas to provide their own, unique, marketable contributions. While potentially valuable in isolation, the aggregation of these additions (and they were, as we shall see, deliberately aggregated in some cases) yields a rat’s nest of complexity.
2. Structured on-ramps. To make OOP appear easier and more accessible, people developed “object-oriented” extensions to existing programming tools and processes. While this made it easy to access the *observable features* of OOP, it made it ironically more difficult to access the *mental shift* needed to take full advantage of what is fundamentally a thought process and problem-solving technique. By fitting the object model into existing systems, technologists doomed it to stay within existing mindsets.

I started giving conference talks based on the concept that this object-oriented stuff was a few simple ideas hiding behind layers of cruft and complexity, and found that these were well-received. Each presented a specific aspect of the overall story: this book is an attempt to bring the whole narrative together.

Organisation of this book

There are three parts to this story. The first, and necessarily the longest, antithesis: a deconstruction of the state of OOP as it exists today. To get to the kernel of a good idea, you have to crack a few nuts. Part one is the agitation that necessarily precedes revolution.

The second part, thesis: a reconstruction of OOP using only the parts that were left over after the antithesis. Part two is the manifesto: once we've seen that the last few decades of status quo haven't been working for us, we can evaluate something that will.

The third, synthesis: a discussion of the ideas from OOP that aren't being provided by today's object systems, and the ideas and problems that OOP doesn't yet address at all. These are the next steps to take to pursue the ideas behind object thinking. Part three is the call to action.

This is not a pure takedown, a suggestion that we have been monotonically doing it wrong for three decades: the antithesis part of this book questions, rejects and destroys a lot of built aspects of OOP, but by no means all of them. And by no means purely the later ones, either: the message is not that Smalltalk was created in some computational garden of Eden and that Sun tasted of the forbidden fruit which doomed us all to Java. Belief in a primaeval wisdom (*urwissenheit*) leads to an uncritical "tradition for tradition's sake" in the same way that belief in primaeval stupidity (*urdummheit*) leads to an uncritical "novelty for novelty's sake".

Rather this is an attempt to find a consistent philosophy, a way of thinking about software, and to find the threads in the narrative and dialectic history of the making of software that are supportive and unsupportive of that way of thinking. Because OOP is supposed to be a *paradigm*, a pattern of thought, and if we want to adopt that paradigm then we have to see how different tools or techniques support, damage, or modify our thoughts.

About the example code

I’ve consciously chosen to use “mainstream”, popular programming languages wherever possible in this book. I have not stuck to any one language, but have used things that most experienced programmers should be able to understand at a glance: Ruby, Python, and JavaScript will be common. Where I’ve used other languages it’s to express a particular historical context (Smalltalk, Erlang and Eiffel will be prevalent here) or to show ideas from certain communities (Haskell or Lisp).

One of the points of this book is that as a cognitive tool, OOP is not specific to any programming language and indeed many of the languages that are billed as OO languages make it (or at least large parts of it) harder. Picking any one language for the sample code would then mean only presenting a subset of object-oriented programming.

Acknowledgements

This book is the result of a long-running research activity, and I hope that any work I have built upon is appropriately cited. Nonetheless, the ideas here are not mine alone (that distinction is reserved for the mistakes), and many conversations online, at conferences, and with colleagues have shaped the way I think about objects. In alphabetical order I would like to pay particular thanks to Steven Baker, Kent Beck, Alan Francis and Daniel Steinberg.