Objects and Unicorns

Mythconceptions in and around the OO World

Kevlin Henney

kevlin@curbralan.com

Introduction

- Object orientation offers a versatile and wide-ranging metaphor and approach to design
  - Now considered a mainstream way of thinking
- But its practice and vocabulary is beset with many less than helpful myths
  - Some have arisen from simplification for teaching, with the loss of the original deeper concept
  - Others have come from a marketing or populist hijacking of fundamental ideas

Myths about Objects

They sought it with thimbles, they sought it with care,
They pursued it with forks and hope;
They threatened its life with a railway-share;
They charmed it with smiles and soup.

Lewis Carroll, *The Hunting of the Snark*

Object Orientation

- The main focus of OO is classes
  - It is really about object roles and capabilities, for which classes provide but one means of expression
- An object's state is central to its raison d'être
  - An object's capabilities are central, and these may or may not require object state
- Inheritance is as important as encapsulation and polymorphism in the philosophy of OO
  - Inheritance is a mechanism: the need to work uniformly with subtypes is the philosophy

Encapsulation

- It is a programming language mechanism
  - It is a design quality
- It is about hiding data representation
  - Private data representation is a consequence — not a cause — of encapsulation
- It means not having to worry yourself with details of data representation or control flow
  - It is about the separation of concerns, not their divorce

Polymorphism

- It is an OO idea or invention
  - There are four forms of polymorphism, of which one is commonly identified with OO
- It is about inheritance and method overriding
  - It is about common interface and substitutable behaviour across different objects, which is not confined to inheritance, static typing or overriding
- It is synonymous with dynamic dispatch
  - Polymorphic binding time can range from compile time to load time to runtime

Design by Contract

- An interface contract is defined in terms of functional pre- and post-conditions
  - For some contracts this assertion model is almost sufficient, but most of the small print and many other contracts need complementary approaches
- Contracts can be independent of the execution model used for object interaction
  - Re-entrancy, concurrency and distribution all require redrafting of sequential, in-process, top–down contracts

Components

- The term is freely interchangeable with 'object'
  - An object is a runtime-created entity; a component is a development- and deployment-time artefact
- They are just very big objects
  - And the design of a city is just a very big house
- Component-based and object-oriented development are mutually exclusive
  - They are complementary

Patterns

- They are concerned with OO design
  - Only the ones about OO are: the rest are concerned with other aspects and forms of design
- They are parameterized units of design
  - A pattern might inform the construction of a generative code framework, but it is not one
- They are naturally connected to the notion of components, as in "patterns and components"
  - Fashion rather than nature puts them into the same phrase: they are quite distinct concepts

Reuse

- It can be planned for and designed in
  - Reuse is, by definition, ad hoc use of something outside the context for which it was designed; there is some notion of adaptation and adjustment
- Using a library or framework is a good example of reuse
  - This is better known as "using a library" or "using a framework", and is an example of use rather than reuse

Myths about Development

"The time has come," the guru said,
"To talk of many things:
Of use – and cases – and object models –
Of processes – and strings –
And why notations unify –
And if we need such things."

(With apologies to Lewis Carroll)

The Practice of Development

- It can be (or should be, or is) like electrical/mechanical/civil/etc engineering
  - Software engineering is a cousin — not a sibling — of each physical engineering discipline
- It can be deskilled thoroughly and effectively
  - At all levels and from all angles it demands skill
- It should be industrialised, so that it fits a production-line/manufacturing model
  - The manufacture is already mostly industrialised

Analysis

- Scenario-based techniques constitute the analyst's strongest tool
  - They provide good coverage and scheduling, but sometimes you need more than stories
- Any system can be analysed, at a high level, in terms of objects
  - Not all problem frames fit objects comfortably, which can lead to skewed models that owe more to synthesis than analysis

Design

- It can (or should) be an ideal abstraction, free of implementation context
  - Without context design is nothing
- It means diagrams
  - Design can mean diagrams, but they may be neither necessary nor sufficient
- It is by necessity up front, and any back-end changes are something else, not 'design'
  - It's all design, whether profactored or refactored

Architecture and Architects

- Architecture is the big boxes and arrows
  - This is PowerPoint marketecture; there is more to architecture than this... one hopes
- By analogy with other architects, software architects should produce blueprints
  - Architects of the built environment do not do this
- Architects focus on non-functional qualities
  - Functional, operational and developmental qualities are of interest — there is hopefully nothing non-functional in a working system

Programming

- It and design are separate activities
  - Programming is not manufacturing: it is a (very detailed) part of design
- It and testing are different activities
  - Testing is a part of programming
- It is no more than coding
  - It is more, and always has been

Java

- It's a great language
  - It is somewhere between 'OK' and 'very good'
- It's a fully object-oriented language
  - Except for the bits that aren't
- Its static type model ensures type safety
  - It has a secure memory model, but most objects are handled in a dynamically typed fashion
- It has no need of operator overloading
  - And Latin had no need of regular punctuation

Outroduction

- It's time to reclaim and preserve some valuable words and ideas
  - Too much vocabulary and too many concepts have been diluted for mass and marketing consumption
  - At the same time, nothing is static; old ideas can be clarified and extended meaningfully
- Behind every myth is a richer and deeper reality trying to get out
  - La vera verita is much more interesting and useful than la false verita