

Re: How to Introduce OO to Structured–Method IT Person?

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2004-11/0447.html>

From: BoonNam Goh (gohbn2000_at_yahoo.com.sg)

Date: 11/08/04

Date: 8 Nov 2004 03:40:19 -0800

Agree with Elliott that should emphasise on OO design as a means to model the real world to reduce complexity and to use objects as the key granule of abstraction.

But before I can reach that state with a person with many years experience with Structured Methods, need to first have him see the benefit of OO in ways he is already familiar with — hence, was thinking of trying some analogy to subroutines and data normalisation.

After he is convinced and willing to learn OO, then hoping it would be easier to get him to learn and apply it.

Sometimes a person new to IT work will find it easier to learn OO than one who already knows other methodologies (eg. Structured) which the person constantly tries to compare and contrast with the OO concept that is new to him.

Universe <> wrote in message news:<tt5uo0tbcs8uf3b0ue35brm2er3fakcdh9@4ax.com>...

> gohbn2000@yahoo.com.sg (BoonNam Goh) wrote:

>

>> *I am looking into how to introduce OO to a Structured–Method Person*

>> *(ie. IT professional who is new to OO but who has done things via*

>> *structured or functional decomposition method).*

>>

>> *Appreciate advice:*

>> *Can I say as a simplification, that one of the first benefits of OO is*

>> *that it allows for "normalization" of methods -- similar to*

>> *logical–data–modelling's (ie. LDM's) normalisation of data?*

>>

>> *Let me explain:*

>> *In LDM, data is normalised to 3rd normal form (3NF) where there will*

>> *be no duplication of data items in the various data entities.*

>>

>> *Similarly, for OO, when we analyse the objects and draw them into a*

>> *class diagram and use GRASP (or other similar methods) to assign*

>> *methods to relevant classes, we are sort of also "normalising" the*

comp.object: Re: How to Introduce OO to Structured–Method IT Person?

- > > *methods so that algorithms only appear in one place in the whole*
- > > *system -- ie. there is no duplication. This automatically makes all*
- > > *algorithms reuseable as "subroutines".*
- > >
- > > *Eg. In functional decomposition, we could have two processes/modules*
- > > *to develop in a system – say, "buy book" and "catalog book". When we*
- > > *do the functional decomposition, we may end up with duplicate*
- > > *algorithms for ValidateISBNCheckDigit in each module and we have to*
- > > *realise that there are such duplicates before we can move them out*
- > > *into a common subroutine.*
- > >
- > > *In the case of OO, we would have naturally put ValidateISBNCheckDigit*
- > > *as a method into a suitable class (eg. "Book") the first time we did*
- > > *an analysis for the first module – say, "buy book". When we do OO*
- > > *analysis of the 2nd module – "Catalog Book", the*
- > > *ValidateISBNCheckDigit method would again naturally go to the same*
- > > *class. Creation and reuse of a "subroutine" would come about*
- > > *naturally.*
- > >
- > > *Does the above sound valid as a sort of simplification to one aspect*
- > > *of OO?*
- >
- > *Yes, but it is akin for many projects to prematurely doing performance*
- > *tuning--attempted before requirements task logic has been completed.*
- >
- > *I suggest that *modelling*, or the same thing *abstracting* –*
- > *eliminating things irrelevant to the requirements and representing or*
- > *enhancing things relevant to project requirements – the project domain*
- > *and requirements should be the starting point of virtually *all**
- > *software development.*
- >
- > *The essence of OO sw engineering is reducing complexity by modelling –*
- > *abstracting* – major requirements relevant role responsibilities, in*
- > *the domain and in the requirements, as a network of collaborating*
- > *object modules.*
- >
- > *"Object–oriented [OO] programming. A program execution*
- > *is regarded as a physical model, simulating the behavior*
- > *of either a real or imaginary part of the world...The*
- > *notion of a physical model should be taken literally."*
- > *~ Computerized physical models*
- > *by Madsen, Moeller–Pederson, Nygaard (co–founder of OO)*
- >
- > *Similarly the essence of Structured sw engineering is reducing*
- > *complexity by modelling – abstracting – major requirements relevant*
- > *data handling processes and entities, in the domain and in the*
- > *requirements, as a network of collaborating event modules.*
- >
- > *There's more to 'em obviously, different secondary things, like "avoid*
- > *goto" for Structured – but these are the core skeletal definitions.*
- >

comp.object: Re: How to Introduce OO to Structured–Method IT Person?

- > *Dependency management (DM), has 2 critical success factors (CSF):*
- > *` minimally loose inter–module coupling*
- > *` maximally higher intra–module cohesion*
- > *DM has a cornucopia of techniques for achieving its CSF's, like:*
- > *– localizing declaration and definition code*
- > *– Law of Demeter (for *both* Structured and OO)*
- > *– minimizing code duplication*
- > *– minimizing interface size*
- > *However applying these DM techniques should be secondary to creating*
- > *the overall system modelling framework, and DM should serve and*
- > *strengthen the operation of the processing model.*
- >
- > *"We will introduce concepts such as information processes and systems*
- > *and discuss abstraction, concepts, classification and composition. The*
- > *framework provides a means to be used when modeling phenomena and*
- > *concepts from the real world, it is thus a fundamental basis for*
- > *object–oriented analysis and design. It is, however, also important*
- > *for implementation, since it provides a means for structuring and*
- > *understanding the objects and patterns generated by a program*
- > *execution. The approach presented in this book is that analysis,*
- > *design and implementation are programming or modeling activities, but*
- > *at different abstraction levels."*
- > *~ Object–Oriented Programming in the BETA Programming Language*
- > *Ole Lehrmann Madsen, Birger Møller–Pedersen, Kristen Nygaard*
- >
- > *[Nygaard is one of the 2 (1st among equals, imo) co–creators of the OO*
- > *sw paradigm.]*
- >
- > *Of course, many multi–layered software system architectures often use*
- > *different modelling paradigms for various layers. I.e. the domain and*
- > *requirements layers may have OO modelling, while the data layer has*
- > *Chen Entity/Relational modelling.*
- >
- > *To repeat, I think its best to begin with a focus on*
- > *modelling/abstraction to reduce complexity in general, with objects*
- > *as the key, or core, granule of system abstraction.*
- >
- > *Modelling in mainstream OO sw engineering has multiple overlapping*
- > *functions. Modelling is the basis for understanding the domain and*
- > *use case requirements. And fortunately, a key thing that makes OO*
- > *modelling a big win, the OO analysis models – class, sequence, state,*
- > *etc – can and should serve as the core skeleton of the system's high*
- > *level – user interface, domain model, and requirements model –*
- > *physical design. Just as Nygaard et al state in the last excerpt.*
- >
- > *This is the way OO's *seamless* design is realized. This means key*
- > *analysis object model entities should be preserved as the skeleton for*
- > *the system's high level physical design.*
- >
- > *Elliott*