

# Re: defining quality of OOA and OOD models

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.object/2005-08/msg00463.html>

---

- *From:* "H. S. Lahman" <[h.lahman@xxxxxxxxxxx](mailto:h.lahman@xxxxxxxxxxx)>
  - *Date:* Sat, 13 Aug 2005 18:02:24 GMT
- 

Responding to Davor...

There are a couple of books on Amazon devoted specifically with OO metrics, many of which reflect some assessment of quality (dependency analysis, benchmarking "accepted" guidelines, etc.). Essentially those metrics are measuring conformance to accepted methodological practice.

Yes... Many of these UML case tools also include some metrics, but I'm looking for something that people actually use in their day-to-day work to evaluate their models "just by looking at them"...

For day-to-day work I think one relies on peer reviews. The peers provide the knowledge base for the methodology and its practices and the reviews install that knowledge base in the development process.

Where the metrics come in is during process improvement as they provide data on the way the process (including peer reviews) /actually/ worked. (We used the metrics during project and increment post mortems.) For example, if the metrics indicated that the peer reviews were not producing the desired conformance during daily work, then the developers need to find a way to improve the peer review process. (Or find an alternative but IME improving the review process is sufficient.)

RA and OOA are quite different things. RA describes the problem (What must be done) while an OOA model describes the solution (How it should be done).

Might be, but I see them as complementary techniques (though not with definition of OOA that you are using) since RA has also evolved quite a bit from just functional requirements. For example, use cases capture and correlate functionality, behavior, and some of concepts that exist

## Re: defining quality of OOA and OOD models

in the problem domain. The very same way conceptual diagrams help us understand and correlate some of functional requirements...

That's fine; use cases are a form of requirements specification. There are even OOR specification approaches that use UML as the notation for specifying requirements. (Mainly because both OOR and OOA need to be independent of the computing space and UML is one of the few general purpose notations that is abstract enough to ensure that.) But that sort of problem /specification/ is still quite different than developing an OOA model software solution.

OOA, OOD, and OOP are just different aspects of solution design. OOA deals with only functional requirements and specifies the solution in a manner that is independent of local computing environment issues. (An

I really like much more that other term that you used before for what you are referring to (Platform-Independent Model (PIM) or something). Using the term "analysis" to mean "design" should, if nothing else, not be used at least to avoid confusion. OOA that I'm referring to is something that should be used to understand and communicate the problem domain, and as such *might* be used as a solid starting point for specifying PIM...

The PIM is an MDA term and it has only recently appeared on the scene. It is semantically equivalent to "OOA model".

I agree that using 'analysis' created confusion compared to an SA/SD context. OTOH, there was clearly a need to provide more step stones to bridge the chasm between customer problem spaces and the computing space. In the SD world they tried to get around the problem by introducing notions like "high level design" and "low level design". But those terms weren't well-defined and the borders were murky. So the OO people decided to emphasize the split and differing concerns by using different terms for the three step stone facets of software design.

The problem was that all the potentially useful names already carried traditional development baggage so they had to pick the lesser of semantic evils. FWIW, I agree that PIM is probably a better name, but OMG had two decades of hindsight to come up with it. Until MDA becomes commonplace in SW development we will be stuck with "OOA Model".

acid test of an OOA model is that it could be unambiguously implemented without change as a manual system.) The OOD model augments the OOA solution by resolving nonfunctional requirements at a strategic level.

## Re: defining quality of OOA and OOD models

I was always having problems with "augmentation" in the sense you are referring to - I have never been able to just augment at any level of analysis or design - some things get cut out, some completely changed, some stay as they are... I don't know much about MDA, and the things might have changed, but I still have to stick with non-MDA development...

But elaboration (the conventional term for your 'augmentation') has been the traditional software development approach for nearly half a century, starting with HIPO charts. One always decomposed higher level, abstract designs by augmenting them with implementation detail to produce a more concrete model of the solution. It was only when translation approaches appeared in the early '80s that one had an alternative through automating the computing space.

The OOP model augments the OOD solution at a tactical level. Each design approach addresses different issues and employs different techniques (e.g., the problem space abstraction that dominates OOA is supplanted by 3GL dependency management during OOP). IOW, each design approach separates different design concerns (e.g., functional vs. nonfunctional requirements, customer vs. computing space views, strategic vs. tactical, etc.) in order to better manage complexity.

So an extremely simple example: Have to make a small simulation that among other things has a "customer queue".

In my OOA model, I would probably have a "customer queue" concept linked to a "customer" concept. In my high-level OOD (which seems to correspond to your OOA model), I would probably realize that I can just use generic software design concept "queue" instead of "customer queue" and link it to a "customer" class. Then I might realize that I don't

Using a generic library class for the queue is a pure OOP tactical decision...

need a "customer" class at all but rather just some "time units" to simulate time differences between customer arrivals... So more detailed OOD model might have "queue" and "time interval" concept... Then when

That is a pure OOA issue. One has to abstract problem space entities to provide an OO solution. In doing so one must decide /which/ entities are relevant to the solution. In a queuing problem things like interarrival

## Re: defining quality of OOA and OOD models

times are intrinsic concepts of the problem space (i.e., queuing theory). So if they will resolve the problem better than a notion of customer, then that is what one abstracts at the OOA level.

```
programming, let's say in Python, this design model concepts "queue"
becomes a Python's list and "time interval" just an integer... Then if
there is a need to optimize, Python's list might become a C array and
Python's Integer might become C byte and both imported in the rest of
Python program... and so on...
```

Fine. A Python program is just a software model at a much lower level of abstraction than either an OOA or OOD model. It deals with a different set of tactical implementation issues. [Not to mention a host of developer issues related to source maintainability when one is so close to the hardware computational models! B-)]

```
So, as you can see, I do not expect OOA to translate to OOD and OOD to
OOP, but I still find that it is useful to model each concept at the
appropriate abstraction level and with a certain purpose. "Customer
queue" from the OOA model completely disappears later on, but its value
for the understanding of the problem in unquestionable... So, I am
curious, how would this simple problem be specified in MDA through
augmentation and translation?
```

As I indicated above, I see the choice of Time Interval vs. Customer as an OOA choice because it reflects how one must abstract the problem space. However, "Simple" examples are deceptive. You have addressed one part of the problem that I would relegate to a particular subsystem...

At the level of queuing theory things like interarrival times are relevant. But the odds are that one is solving a problem for some customer who doesn't know or care about queuing theory (e.g., someone estimating resources needed to man a hotline). Then one may well need abstractions from that problem space like Customer. So the application might well have to have to deal with both views. One handles that by application partitioning of the subject matters (queuing theory vs. hotlines) into subsystems where each subsystem employs its own unique views of the relevant problem space. The high level control would reside in the subsystem that understood Customers while the actual algorithmic grunt work would live in a subsystem dedicated to the queuing theory subject matter. One of the things one does during OOA is to sort out such systems engineering issues. B-)]

```
In the interest of pickiness, I never regard correctness as a quality
criteria. Correctness is just an exit criteria for a design activity.
If the application doesn't work correctly, then it isn't finished yet.
```

## Re: defining quality of OOA and OOD models

Agree. Same could be said about completeness too if we define completeness in terms of correctness, which you just did.

That was not my intent. In my view completeness refers to whether the application addresses all the requirements (correctly or not). The distinction is analogous to verification (the developers built what they intended to build) vs. validation (the developers built what the customer wanted them to build) in software testing.

\*\*\*\*\*

There is nothing wrong with me that could not be cured by a capful of Drano.

H. S. Lahman  
hsl@xxxxxxxxxxxxxxxxxxxxx  
Pathfinder Solutions -- Put MDA to Work  
<http://www.pathfindermda.com>  
blog: <http://pathfinderpeople.blogs.com/hslahman>  
(888)OOA-PATH