

Re: OO Design induces an existential crisis

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2005-07/msg00824.html>

- *From:* "Nick Malik [Microsoft]" <nickmalik@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 18 Jul 2005 08:40:44 -0700
-

"topmind" <topmind@xxxxxxxxxxxxxxxxxxxxx> wrote in message
news:1121550324.682122.32490@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

> I don't necessarily even disagree with them.

It is hard to believe that you would consider agreeing with the principles of OO design, when you have made such a spirited case for calling them pointless.

Some of the most innovative ideas in software development have come from examining a set of principles, in detail, and challenging one of them (while leaving the rest). I would suggest that one of the driving points for the creation of XP was Beck's decision to challenge the long-held notion that "the longer you wait to change a requirement, the more expensive it becomes to change." He didn't challenge the rest. Just that one. He shows that if you challenge that one assumption, you can completely change the conversation about software development.

I ask you to find the single OO principle that you disagree with. "if OO purists would abandon this one, then their code would be more realistic/understandable/useful/agile... whatever..."

- > I just cannot evaluate the
- > change pattern frequencies assumed in the articles because I have
- > insufficient experience with that domain.

I don't buy that. You call yourself an experienced business applications developer. If you have that experience, then you've seen an application go from "great new idea" to "a place to add features" to "the yucky job you give to the new guy." If you deny the existence of code decay, then I would challenge your experience in a business setting. Ask others around you about their experience with applications that lasted more than 5 years of continuous "improvement".

- > Good. Show it improve the change-handling of *business* examples then.

EDRA

<http://www.gotdotnet.com/workspaces/workspace.aspx?id=9c29a963-594e-4e7a-9c45-576198df8058>

Re: OO Design induces an existential crisis

This is a framework, developed on top of Microsoft .Net technologies (but could just as easily be developed on Java) for web service integration points that creates a place to put logic, and different place to put "cross-cutting concerns" like logging, authentication, and routing. This allows these concepts to be changed independently of one another in a way that is pretty good for meeting the principles of both OO design and SOA architecture.

> There must be a good reason why OO training examples roughly have a
> ratio of 25-to-1 in favor of sys software examples over biz examples

Sure... most of the people who add these features to a language work in the field of writing languages... e.g. system software, so they explain it from their viewpoint. Once the business developers learned OO, they looked around and said "the way I learned it is from these sources, so they are good enough for someone else to learn it..." so they didn't create new sources of info. It's a bit of a vicious cycle.

The book I quoted attempts to break this cycle. It was written by a business user who learned OO and then became an architect and now runs a company that trains people on how to use OO. The man's name is Alan Shalloway. His company is Net Objectives. The book is "Design Patterns Explained" but he doesn't just explain design patterns. He explains the transition he made from procedural to bad-OO to good-OO.

> The ambiguity is in the change patterns they assume and test. You
> cannot hide from probability. I can even give change scenarios that is
> more code rework in the device driver examples.

Change is hard and often unpredictable. Other times, it is completely predictable. The reason that folks like to use the "device driver" as an example is that everyone understands the notion of a bit of software that helps the operating system deal with change (in devices). It is a shared concept that an author can build on. It has nothing to do with system software as the only domain for OO problems.

When I write an article (which I do often), I don't have time or space to define every term. I need to rely on concepts that already exist in the reader's mind. I do not know my reader, but I know some things about them. If they are reading one of my articles, they are probably a developer familiar with the Windows environment. Therefore, they already know what a device driver is. So I can use that. I don't have to. I could use something else, but I'd have to take time and space to explain it. That is one more reason that device drivers are cited so frequently. It has nothing to do with that being the only domain for OO.

Seperately, I would posit that probability has very little to do with the concepts of OO or how best to apply them.

You don't create the "interface" only at the points where change will occur, because, as you know, change often occurs somewhere else. You put interface

Re: OO Design induces an existential crisis

points between *concepts* because concepts change at different rates.

When a new "boss" comes into the project and changes things around, the things they change will nearly always line up with one or two concepts. Therefore, you can change the code for those concepts separately from the code for the rest of the system.

For example: In my car, it is fairly easy to get to the battery. It is less easy to get to the alternator. It is downright difficult to get to the air conditioner compressor. Why? It is because this is the order of the frequency that a repair person needs to get to these parts. However, each of these parts can be removed and replaced without messing with too many other parts. You don't have to dismantle the engine to replace the alternator. There is no probability in that notion.

Sure, we could guess that it is more likely that we would need to replace the alternator over the engine... we could even cite statistics to support the notion. But it is less important than it appears. The fact is that the repair of the alternator would usually happen at a different RATE than the repair of the engine. Therefore, there needs to be a way to get at one without causing complications with the other.

Now, in many cars (mine included), you may have to move the alternator out of the way if you are servicing the engine. You don't have to move the engine out of the way to service the alternator. This is where you could use probability to improve the efficiency of your operations. However, the interface exists, regardless of the statistics, because the RATE is different.

While you cannot tell what will change first, being familiar with the business domain will quickly give you a respect for the things that change at different rates.

- > The ambiguity is in the change patterns they assume and test. You
- > cannot hide from probability. I can even give change scenarios that is
- > more code rework in the device driver examples.
- >
- > Nobody has questioned that in such debates that I remember. In other
- > words, nobody has ever disagreed with the suggestion that a given
- > design cannot weather ALL changes better than the competition.

What does "more rework" mean? Is cost a factor of the lines of code? Being a function point counter and a cost wonk, I would emphatically disagree. Cost is a factor of complexity of the requirements. If you shear a design to go from less complex to more complex, there will be changes in many lines of code. This happens regardless of whether you are in the OO or Procedural space. No design can (readably) create an interface for all change, especially change that crosses this complexity boundary.

I would state that, on average, OO code tends to require fewer lines of code to add a feature than procedural. That is anecdotal at best, and you have

Re: OO Design induces an existential crisis

every right to doubt that statement. I cannot put science behind that statement.

However, I would also suggest that the sheer number of lines of code changed is a poor measurement of the cost of a feature. It is entirely possible to change three lines of code in a procedural application, and then require a full regression test of the system and all integration points, while I have seen situations where a developer changed the fundamental structure of part of an OO application and it required only the simplest build–verification–test to validate that the code was sound. This is just one (extreme) example that shows that any measurement for capturing the notion of cost must include more variables than the KLOC count.

>

> If you agree with this, then you must agree with me that probability
> plays a role. That is, we pick the design that reduces the change
> impact of the (estimated) most *common* changes.

Nope. I do not agree. Any time I ask for justification for code interfaces, and the comment comes back as "I am reducing the impact of change", I do my level best not to break out in laughter. As I said before, a good design captures the connections between the "concepts", not the points where a developer (in his "infinite" wisdom) expects change. To say that we know where to expect change is nuts. We aren't the business users, so we don't have the appropriate context. Besides, the average developer isn't smart enough to predict that change. We just need to predict different rates of change. That part is a good bit easier to do.

> It seems that OO'ers tend to be bad proceduralists/DB'ers. Thus,
> perhaps they are abandoning p/r because they suck at it, not because of
> any inherent fault in it.

I could levy the same comment at those who write procedural code. Let's see if we can avoid these little tidbits, OK? They only serve to lower the tone of the discussion.

>>> Are there any OO gurus out there who want to take on custom business
>>> applications? Any?

You asked if there were any gurus out there working in the business apps space. I replied yes (although I wouldn't apply the term "guru" to myself. That is a religious term). You didn't appear to want to take that anywhere. I'm not sure why you asked the question. Are you interested in seeing how a business app can be solved with OO code? There are literally thousands of examples in articles for dozens of magazines.

It appears that you are interested in a comparison of the costs of using OO to solve a problem and the costs of using procedural code to solve the same problem. This wouldn't come from a guru. It would come from a researcher.

I would posit that a comparison can be done, but it would have to take years

Re: OO Design induces an existential crisis

Re: OO Design induces an existential crisis

to study, because you would want to solve the same problem of sufficient complexity, with both mechanisms, and then add changes, over time to both code bases, using a mix of experienced and new developers, in a controlled turn-over environment.

To be fair, the experiment would have to be structured this way because the benefit of one over the other may not be realized during initial implementation. Benefits may only show up during long term maintenance of the code. This could go either way.

That is why few studies exist. They are expensive and time-consuming. Without a requirement in legislation, or an edict from the board of directors, no one has the reason to spend the money for this kind of study.

However, the anecdotal evidence is so powerful that most folks who write code today do so in an object-oriented language. Millions of developers are improving their skills on OO development every day, because they've seen the benefit of doing so. Sure... some of that it top-down. Your boss tells you to write in Java or C#. However, your boss can dictate the language, but not the technique. The improvement in technique is because smart people are discussing principles and ideas, and that leads to innovation. This is happening, but it takes time.

In this climate, there is very little motivation to perform this kind of study.

>> You have convinced me of one thing: you have never seen OO done
>> correctly.
>> Therefore you are convinced that it cannot be done.
>
> No! I am only saying that public evidence does not exist.

Are you saying that you *have* seen a business OO application that was well constructed and met its objectives of keeping costs low? In that case, your viewpoint is even more perplexing.

> Yet companies
> are spending zillions on OO. So quick to spend without any real science
> or comparative analysis?

Are companies really spending zillions on OO? Companies are spending zillions on customer software development. That is the problem. They should write far less code than they do. For most companies, they don't give a hoot if the technology under the covers is OO or PP or QQ. They care about results.

This is why the study may have limited value. It is a technical choice, not a business one. To be honest, even if a study did come out, I'd be surprised if it showed there to be a huge cost difference. Probably only about 10% of the cost of coding would be gained or lost either way. Considering the fact that coding makes up only about a third of the cost of

Re: OO Design induces an existential crisis

custom software, we are haggling over 3% to 4% of the budget. Worth haggling over, but not so much that business even cares. They can eat 20% of their costs in poor requirements gathering, 25% in poor project management, and 20% in poor testing. 3% is not even on the radar.

- > The format should look something like:
- >
- > Here is the OO code:
- >
- > blah blah blah...
- >
- > Here is the procedural/relational equiv. code:
- >
- > blah blah blah...
- >
- > Here are 5 change scenarios:
- >
- > 1. asdfasf...
- > 2. adfasdfasdfs...
- > 3. asdsdf.....
- > 4....
- > 5....
- > (with possible frequency estimates)
- >
- > Here are the scores for each scenario:
- >
- > Number of lines changed: ...
- > Number of blocks changed: ...
- > Number of named-units changed: ...
- >
- > Got Science?

Got time? I don't. I have a job to do.

Why not write up your experiment ideas and send them to college computer science departments around the world. Someone will bite, I assure you. Every year, graduate students are desperately seeking an interesting topic for their thesis. I'm sure that some of them would love to take this on.

---- Nick Malik [Microsoft]
MCSD, CFPS, Certified Scrummaster
<http://blogs.msdn.com/nickmalik>

Disclaimer: Opinions expressed in this forum are my own, and not representative of my employer.

I do not answer questions on behalf of my employer. I'm just a programmer helping programmers.

- **Follow-Ups:**
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: topmind

- **References:**
 - ◆ **OO Design induces an existential crisis**
 - ◇ From: kj
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: Rick Elbers
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: kj
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: Rich MacDonald
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: topmind
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: Nick Malik [Microsoft]
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: topmind
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: Nick Malik [Microsoft]
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: topmind
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: Nick Malik [Microsoft]
 - ◆ **Re: OO Design induces an existential crisis**
 - ◇ From: topmind

- Prev by Date: **Re: OOP/OOD Philosophy**
- Next by Date: **Re: Measuring OO Reuse**
- Previous by thread: **Re: OO Design induces an existential crisis**
- Next by thread: **Re: OO Design induces an existential crisis**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**