

Re: Liskov Substitution Principle and Abstract Factories

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2005-01/0794.html>

From: Mark Nicholls (nicholls.mark_at_mtvne.com)

Date: 01/14/05

Date: Fri, 14 Jan 2005 14:40:58 -0000

"Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de> wrote in message
news:pg92gmhphg0o.yv7wsr04v0lo\$.dlg@40tude.net...

> On Thu, 13 Jan 2005 18:17:36 -0000, Mark Nicholls wrote:

>

>> "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de> wrote in message

>> news:fwowg4yqpybz.1g0qhrqzswing.dlg@40tude.net...

>>> On Thu, 13 Jan 2005 15:36:31 -0000, Mark Nicholls wrote:

>>>

>>>> "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de> wrote in message

>>>> news:1bnewz2zpx0o.611uno3swovd\$.dlg@40tude.net...

>>>>> On Thu, 13 Jan 2005 11:17:13 -0000, Mark Nicholls wrote:

>>>>>

>>>>> OO is not equivalent to prefix notation. To put it clear:

>>>>>

>>>>> F is a method of X = the subroutine F has at least one dispatching

>>>>> parameter or result. Period.

>>>>

>>>> "one dispatching parameter" ?

>>>>

>>> [in C++ only one 1st parameter of a virtual function is dispatching.

>>> Virtual functions dispatch when called (on a reference X.f() or pointer

>>> X->f()). The actual type is determined by the vtab pointer. vtab is the

>>> dispatching table of the corresponding polymorphic function. Because

C++

>>> has single dispatch, its dispatching tables are vectors. In general

case

>>> they are n-dimensional matrices, where n is the number of dispatching

>>> parameters.]

>>>

>>> this is what I need to absorb.....to be I see v-tables...your saying in

>>> general they are matrices?

>>>

>>> clunk, clunk, churn churn.

>>>

>>> so

comp.object: Re: Liskov Substitution Principle and Abstract Factories

> > *the 'function' call*
> >
> > *void foo(int, char, double)*
> >
> > *is dispatched via a three dimensional set of function pointers?*
>
> *Yes. int, char, double have type tags which are used as indices for the*
> *matrix, if you mean how that could be implemented.*

yes....good.

>
> > *then how do you get to methods?*
> >
> > *in C++*
> >
> > *void foo(int, char, double)*
> >
> > *maps to*
> >
> > *int::foo(char, double)*
> >
> > *and in a sense that would seem central to OO modeling i.e. you attach*
> > *methods to classes, not to cross products of classes.*
>
> *But that's obviously wrong. Methods are attached to tuples of types not to*
> *singular types. Otherwise you will never get arithmetic operations,*
> *assignment, equality etc right.*

now wait.

OK from your context you claim this is obviously wrong....that's fine....but what I am saying is it does not seem consistent with OO.

```
void Dog.Bark() // method on dog  
void Dog.Bark(string what) // method in dog ? or dog cross product  
string ?
```

I have observed this before and it has worried me but it seems not to worry you?

>
> > *how do you do a class diagram if the methods do not belong to a specific*
> > *class, but to a cross product of them?*
>
> *I do not put methods in types anyway when I draw a diagram. To it is*
> *relationships between types is of the major interests. Then you can still*
> *place a method in a type box if it dispatches on this type, just do not*
> *forget that it may also appear in some other boxes.*

how utterly bizarre.....not bad or wrong....just bizarre.

to me this is ER modelling and overlaying a functional model (with indirection via function pointers) on top.

Bigger, better and more general.....but not what most people would call OO.

In fact bizarrely the other day I was doodling some class diagrams, I also do not put methods on normally, but I was perplexed by the coupling that interfaces were causing in some software (its the compilers fault not mine), and I ended up with the methods in boxes connected to the classes....usually multiple boxes.

>
> >>>>>> *if it does that you *know* (well almost) that P is a member of that*
> >>>>>> *set.....isn't that the basis of design by contract.*
> >>>>>>
> >>>>>> *It is, but how do I know it?*
> >>>>>>
> >>>>>> *many contracts are provable.*
> >>>>>> *most software is statistically 'provable'...i.e. testing. The compiler does*
> >>>>>> *generally prove alot.*
> >>>>>> *So if I can say*
> >>>>>>
> >>>>>> *$p(A \text{ subtype } B) = 0.99$*
> >>>>>>
> >>>>>> *I can very dubiously say*
> >>>>>>
> >>>>>> *$p(\text{LSP}(A,B,\{\text{programs containing } A\}) = \text{true}) = \text{approx } 0.99$ (I accept*
> >>>>>> *this is highly dubious, there are all sorts of assumptions).*
> >>>>>>
> >>>>>> *First, what is p? Definitely it is not Pr (probability). And if that were*
> >>>>>> *Pr, then what could it mean? You know any Pr not equal to 1 guaranties*
> >>>>>> *nothing. However, statistic cannot be applied here. It is not a stochastic*
> >>>>>> *thing. The program is either wrong or not (conditionally to the*
> >>>>>> *requirements given).*
> >>>>>> *So a bug is either here or not. Consider a statistical*
> >>>>>> *experiment: you run the program 1000 times with the same data and see*
> >>>>>> *how*
> >>>>>> *many times the bug shows itself. Nonsense. The only random thing lies*
> >>>>>> *outside, in our perception of a program and its validness. That can*
> >>>>>> *be*
> >>>>>> *considered as random variables. But that has nothing to do with the*
> >>>>>> *program*
> >>>>>> *itself. It is psychotherapy, at best.*
> >>>>>>
> >>>>>> *If we cannot logically prove correctness, then we can only*

statistically

> >>> *define it.*

> >>

> >> *How? To define something statistically you have to present the space of*

> >> *elementary outcomes. What are elementary outcomes? Failed program runs?*

> >

> > *it could be the expected value of 'success' over the cross product of*

> > *inputs.*

> >

> > *success would be if expected behaviour = contracted behaviour.*

>

> *Those are not random! For each input the program either fails or not, it*

is

> *deterministic.*

If you do not know (or cannot formally analyse) the implementation of the program, then the program is a random variable.

Is the weather deterministic?

or just so complex as to not submit to complete analysis.

Is the sun going to be shining tomorrow? do I need to formally analyse the physics of the sun to answer yes?

Isn't it a bit like schrodinger's cat (though I do not claim to understand it or even know if it exists.....without looking.....aaaahhhhh).

That's what engineering is about.....and if you believe quantum physics...the universe as well.....but I'll leave that to people that understand it.

>

> > *you would then have to throw some spurious assumptions at this to enable you*

> > *to approximate that....but that's statistics.*

> >

> > *If I build a bridge I am not sure it will stand, but I may be 99.999% sure*

> > *it will.....and then the wind blows in a specific direction, at a specific*

> > *speed and we discover that it resonates, and falls down.*

>

> *The laws of physics describing behavior of a construction have statistical nature. This is why one can talk about Pr (bridge will stand). The laws of*

> *Boolean algebra have nothing to do with statistics. So Pr (program is*

> *valid) is sheer nonsense. It is like to talks about Pr (2+2=4).*

not until you evaluate the implementation

P(s = 4)

comp.object: Re: Liskov Substitution Principle and Abstract Factories

S member of domain { 1+1, 2+1, 3+1 }

s is a member of S with even distribution. (assumption!).

P(s = 4) ?

answer 1/3

P(that this instance of outlook express will actually send this post) = 0.99 (approx).

I see no problem.

>
> >>>...that's the difference between maths and physics/engineering.
> >>>
> >>> not ideal....no....but this is the reality.
> >>
> >> You cannot apply any theory without reasonable justification.
> >
> > I have no model as yet.
> >
> > naively
> >
> > if a program worked yesterday, and last week, and for the last 3 years,
you
> > would expect it to work tomorrow.
>
> On different data? Come on, it is like to expect that if test A passed
then
> the test B would too. OK, it is in human's nature to hope in that, but it
> has nothing to do with the reality.

yes it does.

If you have a calculator and you type in 10,000 different combination of x + y....

will the 10,001 work..

you claim that you do not know.

I claim P(10,001 works) is pretty high probably about 0.9999 based on a naive model in my head of calculators and how they work.

Not everything falls to pure logic, we know that, we can logically prove that.....for the rest we have to guess.....and it's called statistics.

>
> > It is not proof, but you do this every day, you have a model in your head

comp.object: Re: Liskov Substitution Principle and Abstract Factories

- > > *that says so.*
- >
- > *Right. This is what I meant. The randomness lies in our heads, not in the*
- > *software. So the "theory" describes not the distribution of faults, but*
- the
- > *distribution of our beliefs in the software. Good to fake up our*
- customers,
- > *but no other use. Just psychotherapy, as I said before.*

But are not my beliefs largely consistent with observations made in the real world (hopefully).

the 'theory' in my head is based on beliefs yes...but these beliefs are not completely irrational and worthless.

to say $P(X = x) = 0.99$ is not worthless just because 1 time in a hundred it is false.

Even $P(P(X = x) = 0.99)$...which is what you are questioning....i.e. the probability that the model does indeed reflect some sort of reality.

(actually this is largely pertinent to the nature of legal trials in the UK.....'expert witnesses' have oft claimed things like $P(\text{second child dies of cot death}) = 0.00000001$...i.e. guilty of murder.....only to find that the probability that the model is in fact flawed is far less....and in hindsight closer to 1.....so the question becomes a conditional one $P(\text{second child dies of cot death} / \text{experts model actually is correct})$...which is a much bigger number).....i.e. the opinion of an expert cannot be worth more than your opinion of experts in general....which aint much.

- >
- > > *If you can produce a reasonable subset of the space of inputs, and*
- relate
- > > *those to the set of inputs likely to happen in a given program, then you*
- > > *have the basis for a statistical indicator.*
- >
- > *Huh. To do this you have to exactly know how the program behaves. But this*
- > *is what you are trying to describe! Otherwise, I would claim that the*
- > *program behaves independently on different inputs. But how can you imply*
- > *any program behavior on untested inputs? That's the first question. The*
- > *second question is how do you know that the produced subset of inputs*
- > *reflects the customer's set of inputs? We know too well that all programs*
- > *are used in a way nobody (either programmer or customer) predicted during*
- > *development.*

true....I'm not claiming it is easy...or even that sensible....just a possibility when all else fails.

take the calculator example...

comp.object: Re: Liskov Substitution Principle and Abstract Factories

take a 10,000 calculators at random and run the same test i.e. type 10,000 random $x+y$ combinations, and verify the answer and then verify that the calculator is working and work out the distribution of the 10,001 test working on the 10,001 calculator against the 10,001 test on the 10,000 and the results from the previous tests).

you now have a statistical model (possibly flawed) that will give you the probability of taking a calculator at random and type $1+1$ and getting 2.....not proof, but I would expect that it would actually be highly accurate.....in fact we could statistically test the accuracy of this method and apply it to other things....e.g. washing machines.

OK, programs are far more complex...but they are not unpredictable....if they were we would not use any of it, as none of it has actually been proved in the manner you expect, and we should have no rational belief to expect it to work.

yet you do.

>
> >>> *I cannot verify the implementation without looking at it!*
> >>
> >> *But contract is less than semantics. Otherwise, there were no need in*
> >> *programmers!*
> >
> > *I can verify that a set of contracts are consistent.*
> >
> > *I need to look at the implementations to verify that the implementations*
> > *satisfy the contracts....thus can I not claim that the set as a whole*
> > *satisfy the contract.*
>
> *OK, but DbC assumes that you start with contracts. Implementations come*
> *afterwards. But the problems with substitutability appear long before*
that.
> *You put down the contracts, everything seems to be OK, and then oops,*
> *circle cannot be resized!*

OK.

>
> >> *This is what polymorphic programs are. You have some polymorphic Pi*
defined
> >> *on the class A'Class. The class includes A, B and all other derived*
types.
> >> *P2 is defined on A'Class. It has an implementation for A and another*
for B.
> >> *P1 is also polymorphic, but we didn't override it for B, so B uses the*
> >> *implementation given for A.*
> >
> > *I think I'm catching on, are you viewing P as a polymorphic type as well*
as

comp.object: Re: Liskov Substitution Principle and Abstract Factories

> > A?
>
> *The polymorphic type is A'Class = { x | x in descendant of A }. In the
> terminology I use polymorphic type = class.
>
> Virtual = dispatching = dynamically polymorphic P1 and P2 are defined on
> A'Class, not on A or B. A::P1 and B::P1i are parts of P1.*

oh no we've been here before, and I didn't understand then.

Ok is it the equivalent as me saying.....

interface/base class IWindow

P(IWindow)....i.e. P is defined in terms of IWindow

but MSWindow is a subtype(!) of IWindow, LinuxWindow is a subtype(!) of window.

>
> >>>> *Nope. char is not substitutable for int. Consider a program that
reads ints
> >>>> from a file. Substitute char and you will have range error. It is a
common
> >>>> error. To judge about substitutability you have consider mappings
> >>>> char<->int and the parameter modes of each given method (that
includes the
> >>>> results as well).
> >>>>
> >>>> ! aren't you just saying
> >>>>
> >>>> LSP(int,char, {set of programs using int}) = false
> >>>> LSP(char,int, {set of programs using char}) = false as well!
> >>>>
> >>>> um, could we define a char in such a way that it would though.
> >>>>
> >> *It would useless then. But for a given set of int-programs we could
have
> >> char substitutable. We just cannot do it for all int-programs,
> >> indiscriminately.
> >>
> > not with this, can we not define char, and the set of P operating on it,
> > s.t. we could substitute int.
> >>
> > heres a start
> >>
> > P1 = {}.....success!
> >>
> > P2;
> >>
> > void Print()**

```
> > {
> > char c = 1;
> > print c;
> > }
> >
> > is that not success;
> >
> > I have a set of 2 programs.
> >
> > if that worked I could probably create a whole load of programs by
> > induction.
> > Find an orthogonal basis for the behaviour of those programs, and I have
an
> > n dimensions space.
> >
> > but now your going to tell me that P2 doesn't work.
>
> It works, but already + will not:
>
> char c = 1;
>
> c = c + 256;
```

I hadn't defined +

and if I did I could leave things like $c + 256$ undefined!.....though quickly I expect we get a very weak contract.

```
>
> And yes you can present a set of working  $P_i$ , but it would not help.
Because
> let somebody have written  $P$ , is  $P$  in  $\{P_i\}$ ? It is undecidable.
```

If it only contains orthogonal elements (we need some sort of way of modelling this, and it may be impossible) of the basis set then it must belong to the set.

i.e. is the vector (2,3) a 2 dimensional vector.

if and only if we can express it in the form $x(1,0) + y(0,1)$

$(2,3) = 2(1,0) + 3(0,1)$yes!

```
>
> In my perception the very idea of LSP was to provide a framework for
> program construction which would not let us leave  $\{P_i\}$ . So were the
> definition of how to create new types, the statements about pre- and
> postconditions etc.
```

that's fine.

comp.object: Re: Liskov Substitution Principle and Abstract Factories

>
> >>> *it may still be of practical use.....if the problem of identifying the set*
> >>> *of non halting substitutions is not halting.*
> >>
> >> *Yes we should approach it from the other end, constructively, step by step.*
> >> *Everything that is provable should flow into the language design in the*
> >> *form of language constructs. The unknown rest will always be, but more*
> and
> >> *more routine software design will fall into your safe set S. That's the*
> >> *program for the future.*
> >>
> >
> > *It's unnerving like incompleteness.....(which I also fail to understand*
> > *to the point of almost not believing).*
> >
> > *but*
> >
> > *LSP(A,B,{}) = true....so I know something.*
>
> *Compare, I have to implement sine. Well, $\sin(0) = 0$. I know something.*
> *Moreover I know $\sin(\pi) = 0$. Does it help me much?*
>

It's getting better....

$\sin(n*\pi) = 0$it's getting really good now.