

Comment on Justification



[scott frye](#) 1 posts since

Jul 28, 2005

I was very interested in your article and you make a lot of good points. I think that concept of LOP is wonderful but some of your justifications for it left me with some questions.

While reading the article, a number of questions came to me....

“I’m talking about the limitations of programming which force the programmer to think like the computer rather than having the computer think more like the programmer.”

Is there some advantage to thinking like a computer? Most of the people I know that program are far better problems solvers than those that don’t program. Does thinking like a computer help break down a problem and see sides of it that we didn’t see when we first were introduced to the problem?

“To understand what Language Oriented Programming is, let’s first take a look at today’s mainstream programming. It goes something like this:

Think: You have a task to program, so you form a conceptual model in your head about how to solve the problem.

Choose: You choose some general-purpose language (such as Java or C++) for writing the solution.

Program: You write the solution by performing a difficult mapping of your conceptual model into the programming language.”

Is this process actually a cycle instead of a linear process? Are they discrete steps or do they in fact overlap? What exactly is going on in the programming step? Why is it taking so long? Is it because the programmer is trying to figure out how to translate the code or is there a good deal of thinking and exploration of the problem space taking place in this step?

“Now, the Program step is much less of a bottleneck because the DSLs make it much easier to translate the problem into something the computer can understand”

This is only true if the only action that takes place during the programming step is translation of the problem into code.

“For me, the most serious problem is that there is a very long gap between when I know exactly how to solve a problem and when I have successfully communicated this solution to the computer as a program.”

Is this the bottle neck of the majority of programmers? Or is it only a problem when implementing a problem type that have been solved many time in a similar environment? i.e. seasoned programmers.

“For example, today a good deal of development time is spent on object-oriented design (OOD). This is actually a fairly creative process where the programmer expresses classes, hierarchies, relationships, and such. ... The process of OOD is necessary because these classes and methods are the only abstractions that object-oriented languages understand. It seems like it is necessary and creative, but with Language Oriented Programming, OOD is not needed at all.”

My understanding (correct me if I'm in error) is that OOP is about making classes and methods that the object oriented languages understand but OOD is about applying a design technique to better understand the problem space. Specifically the objects and how they interact. Why is this no longer needed in LOR? Does LOR provide a different technique for analyzing the problem?

“We were taught that computers are modeled after the Turing machine, and so they ‘think’ in terms of sets of instructions. But this view of programming is flawed. It confuses the means of programming with the goal.”

I would be interested in a more detailed explanation of how the Turing machine model “confuses the means of programming with the goal” I thought the Turing machine model allowed us to determine the solvability of a problem and the required steps to solve that problem. The fact that it was easily programmable, I thought, was a useful side effect of this mathematical representation of problems that was developed before modern computers.

When I have a problem to solve, I think of the solution in my head. This solution is represented in words, notions, concepts, thoughts, or whatever you want to call them.

What about problems where the solution is not immediately evident? Or very large problems like implementing an ERP system where the solution has a lot of complexity deriving, not from difficult isolated problems but from the interaction of many simple problems?

“This is the main reason I think programmers should have the freedom to create their own languages—so they can express solutions in more natural forms.”

I frequently will gain insight into a problem by resolving it in another language or expressing it in a different form than the one that is “natural”.

- scott frye



[Andrew Eisenberg](#) 4 posts since

May 26, 2005 1. **Re: Comment on Justification** Aug 14, 2005 2:45 AM

I think the major argument to take out of the article is that different types of problems are best expressed in different languages, as are their solutions.

Eg, a circuit is best described schematically, while a mathematical concept can often best be described using an equation.

What the article argues is that it is "unnatural" if we are forced to think about both using java syntax.



[Fernando Tareco](#) 3 posts since

Sep 21, 2005 2. **Re: Comment on Justification** Sep 21, 2005 3:44 PM

I don't see LOP as a programming alternative to address complex problems solving, but rather as an increase in development productivity and software understandability.

The main idea is to reduce the learning curve by raising the abstraction level when you have to deal with an existing architecture or system. The DSL will perform for you the mapping form the functional concepts and that architecture components.

The way I see it, in the future a DSL will be another delivery when we develop a new system or architecture. In this phase we can use a general purpose language or a DSL oriented for developing that category of architectures or systems. Meanwhile, someone will be in charge of translating to a DSL the mapping between the domain functional concepts and the new architecture artifacts.

When this is done, the new developers dont need to have a profound knowledge of the new system to perform its maintenance or evolution.