By John Sonmez
July 15, 2013

Understanding The Problem Domain Is The Hardest Part Of Programming

What is the hardest thing about writing code?

There are many common answers to this question:

- Learning a new technology
- Naming things
- Testing your code
- Debugging
- Fixing bugs
- Making software maintainable

The list goes on and on.

But as I reflect back on my programming career, and I've conversed with many new programmers that are learning the craft, I've found the single hardest thing about programming is learning the problem domain.

A familiar problem

In a good portion of my Pluralsight courses I show the viewer how to build the "Protein Tracker" application.

I am often asked why I keep demonstrating how to build the same simple application over and over again in each of my courses.

The answer is "familiarity."

When I first started using the Protein Tracker example in my Android course, I was just looking for a simple example of an application that could be easily understood and implemented.

The idea behind the Protein Tracker application is that it allows a user to set a goal for the amount of protein to consume in a day. The user can add protein amounts which are added to a total protein count that is tracked for that user.

Very simple functionality, easily explainable, but most importantly, easily understood.

What I found with this simple application was that because it was so easy to understand, the focus was taken off of the problem domain and put instead on the technology.

Not only this, but as I reused this same exact application for teaching a variety of different technologies, it served as a reference problem domain that didn't have to be re-learned, and provided a way to compare and contrast different technologies—I was getting this teaching mechanism for free if a viewer had already watched one of my other Pluralsight courses.

By creating a familiar problem domain, I found that both the tasks of me teaching a new technology and the viewer learning that technology were much easier, because it is very difficult to learn more than one thing at once.

So what am I trying to say here?

Simply that by taking away the problem domain, or making it so trivial that it is easily understood, I am able to make both teaching and learning easier.

Why problem domains are hard

Have you ever tried to put together a jigsaw puzzle that didn't have any picture on it? How about one like this one, that has a very similar pattern repeated on it and is double-sided?

The reason why puzzles like this one are so hard, is because you can't really see what you are trying to build very clearly. Normally when you put together a jigsaw puzzle you follow steps that might look something like this:

1. Figure out what the major components of the picture are
2. Sort the pieces by color or component
3. Put together all the border pieces

4. Put together each component of the picture from the piles you created

This all breaks down when you don't have a picture with clear components that you can identify.

The same thing happens when writing code. Writing code is a lot like putting together a jigsaw puzzle. We put together code with the purpose of building components that we have taken out of the "bigger picture" of the problem domain.

The big issue is that many problem domains are like a puzzle with a blurry picture or no picture at all.

The real world is a messy place. Many of the problem domains we face as programmers are difficult to understand and look completely different depending on your viewpoint.

As programmers, we also are often not given complete information about the problem domain, so we don't even have the information we need to understand it.

Just try and read the famous Domain Driven Design book and you'll quickly see how complicated and difficult problem domains can be. *(Great book by the way, although you may have to read it twice or three times—I certainly did.)*

Programming is easy if you understand the problem domain

A long time ago, I worked for Hewlett Packard writing software for multi-function printers.

Most of the work at the time was basic waterfall development. There wasn't much Agile happening there—at least at the time I was there.

There was however something really interesting about the waterfall approach and the extreme amount of specification that was done before anything was built—it was very easy to write the code for a feature.

I remember writing a tab control for the user interface of a printer and having the complete pixel perfect specs handed to me before I began to write any code. I was also given all the possible use cases and told exactly how it should function and what it should do under just about every circumstance.

Guess how easy it was to write the code to produce this tab control? Super easy.

As much as I frown upon this approach for software development today, there is something interesting to think about here.

I was essentially given the entire problem domain in the form of a spec that was clear and unambiguous. **I was easily able to learn that problem domain and because of it, I was able to write the code very easily as well.**

Perhaps you have had a similar experience, not necessarily working on a waterfall project where you were given the spec, but perhaps on an Agile project where you took the time to clearly understand the problem domain before writing any code.

I've spent days trying to implement a feature only to finally go back and talk to a product owner and hash out completely how something should work and why it should work in a particular way, only to

go back to my desk and crank out the code in a matter of hours.

The more and more I write code, the more I learn that understanding the problem is the most critical piece to the equation. **It is very difficult to solve a problem before you know the question. It's like buzzing in on Jeopardy before you hear the clue and shouting out random questions.**

What can you do about it?

If understanding the problem domain is the hardest part of programming and you want to make programming easier, you can do one of two things:

1. Make the problem domain easier
2. Get better at understanding the problem domain

You can often make the problem domain easier by cutting out cases and narrowing your focus to a particular part of the problem.

What I mean by this is that it is often beneficial to take a part of the problem and fully understand that part before expanding the problem domain.

Games are really good at this. Look at most games today and you'll find that you start with a very small problem domain. The first level is usually a tutorial that has a basic set of things you can do so that you don't get overwhelmed. But, as you advance through the levels, you usually find they get harder and introduce new concepts that build gradually on what you know, until you understand a pretty large problem domain. (Starcraft is really good at this.)

The other choice is to become better at understanding problem domains. As developers, we tend to think that sitting down and talking to customers or business people who know about the problem

domain is a waste of time. **It is easy to fall into the trap of thinking you understand enough of the problem to get started coding it.** Best to resist the temptation to "not waste anymore time talking" and make sure you understand a problem inside and out before you try and solve it with code. It is much more expensive and time consuming to do things over than it is to do them right the first time. I learn this lesson the hard way time and time again.

Quick update on my new product

I'm still not ready to unveil exactly what I am building, but I do have an active mailing list where you can sign up to find out when I release the product I'm working on to help developers get better career opportunities and market themselves.

So many developers don't realize **how much of an impact marketing themselves and branding can have on their opportunities.** I'm hoping to help developers learn not only how valuable marketing and branding is, but how to do it most effectively.

If you are interested, please sign up. (I won't spam you.)

ABOUT THE AUTHOR

John Sonmez

John Sonmez is the founder of Simple Programmer and a life coach for software developers. He is the best selling author of the book "Soft Skills: The Software Developer's Life Manual."

Related Posts

Productivity is Not Enough. Be Proactive

JUN 19, 2020 / BY DANILA PETROVA

The Effects of COVID-19 on the Web Development Industry

JUN 17, 2020 / BY AARON CHICHIOCO

Your Ultimate Guide To Choosing the Best Test Management Tool

JUN 15, 2020 / BY PATRICK PANUNCILLON

5 Reasons You Might Fail to Become a Software Developer

JUN 12, 2020 / BY ALISSA ZUCKER

Finding New Clients as a Remote Programmer During the Pandemic

JUN 10, 2020 / BY PRATIK DHOLAKIYA

Franklin Method: How To Learn Programming Properly

JUN 08, 2020 / BY CONNIE BENTON

Grow Your Instagram Account To Grow Your Programming Career

JUN 05, 2020 / BY CHRIS FLEGUEL

How APIs Help Developers Build Apps With Enhanced Functionalities

JUN 03, 2020 / BY SOURODIP BISWAS

2 Comments Simple Programmer Disqus' Privacy Policy 1 Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

FrenkyB • 7 years ago

If you add something new to something you already know it's really much easier that in case everything is new. You are completely right on that one. Good idea about using same concept (protein application) for teaching different technologies.

Reply • Share ›

jsonmez Mod FrenkyB • 7 years ago

Thanks. It took me a while to figure it out, but I use the same technique for teaching myself new things as well.

Reply • Share ›

Subscribe Add Disqus to your site Do Not Sell My Data DISQUS

←Previous post Next post→

FREE Email Course

5 Learning Mistakes
Software Developers Make

ENROLL TODAY!

Your first name...

Your email address...

I'm In!

TRENDING | POPULAR | RECENT

Franklin Method: How To Learn Programming Properly
June 8, 2020

Finding New Clients as a Remote Programmer During the Pandemic
June 10, 2020

5 Reasons You Might Fail to Become a Software Developer
June 12, 2020

Your Ultimate Guide To Choosing the Best Test Management Tool

June 15, 2020

The Effects of COVID-19 on the Web Development Industry
June 17, 2020

Productivity is Not Enough. Be Proactive
June 19, 2020

FREE BLOGGING COURSE