TREXIN INSIGHT PAPER
August 2017

BEING A MINDFUL DEVELOPER

Taking the time to listen, learn and understand.

Not that long ago, I suffered from the same affliction that many developers share. I felt that if I wasn't actually writing code, actually typing into IDE, I wasn't doing real work. The only things that mattered were writing code or debugging it. I was wrong. In general, as developers, we need to stop and think more. We need to be mindful developers.

UNDERSTANDING REQUIREMENTS

If we don't understand what we're supposed to be building, it won't matter how good our design is or how beautiful our code is. We will wind up building the wrong thing. Before anything else, we need to stop and understand what we are building. We need to understand the requirements. It is the most important thing that we do. It is also the most difficult.

The basic question we should ask is "who wants to do what with which objects?" Everything starts there. Who are the actors? What are the actions? What are the affected objects? We can then consider what restrictions and dependencies are involved.

WHO IS OUR CUSTOMER?

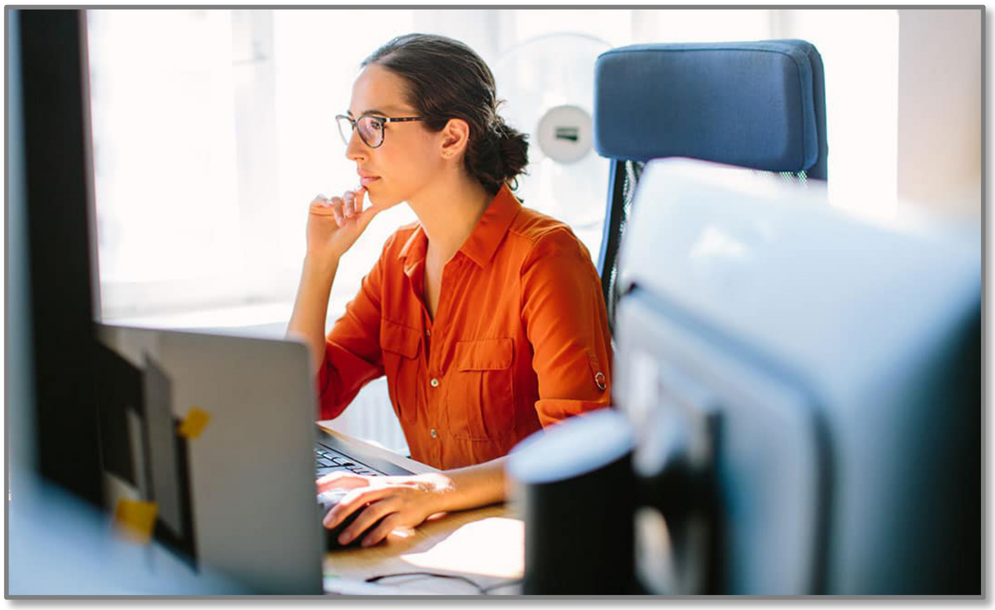
To get a deeper understanding of our requirements, we need to understand who is our customer. Who is going to use our software? Who are we going to support going forward? If we want to follow the Design Thinking approach, we must not only understand but also empathize with our customer.

WHAT IS OUR GOAL?

Before designing or coding our solution, we need to understand our end goal. What is it that we are attempting to do? For example, are we building a Patient Entry screen, building a next generation Electronic Medical Records system, or transforming how doctors treat their patients?

DESIGNING THE SOLUTION

Before we start coding, we need to design what we're building. Writing code without a design is like leaving our house without knowing where we're going. We're unlikely to get to our destination.



We typically start designing at the highest level, usually the process or service level. We should consider what processes will be required. What messages or events will the processes receive or generate? What will be included in those messages and events? It may be useful to draw a picture using pen and paper (or a white board) to capture that information.

Once we understand that level, we then go to the class level and begin to think about how each process or service is composed. What classes will be required? What will each class do? What methods and instance variables will they contain? It's a very similar process to the one used at the previous level. It may be useful to draw another picture.

WRITING BEAUTIFUL CODE

We should always strive to write beautiful code. We all know it when we see it. Beautiful code is uncluttered and elegant. It only does what it needs to do. The class, method, and variable names all reflect their true purpose. They are obvious. If we have a class name like "MyController", we should probably rethink our design. What is the class controlling? What are its responsibilities? We need to understand that in order to write beautiful code.

THE THREE MOST IMPORTANT SOFTWARE DEVELOPMENT PRINCIPLES

When designing and writing code, there are three basic software principles that will make us better software developers: DRY, YAGNI, and KIS. If we follow only these three principles, we will be competent developers.

DRY: DON'T REPEAT YOURSELF

If we find ourselves copying and pasting code, we should stop and consider writing a method, class, or library that we can reuse. Copying and pasting code is evil. We should never do it.

YAGNI: YOU AIN'T GONNA NEED IT

We should not introduce functionality that has not been requested. We'll likely write code (and increase complexity) without ever needing it. We should not use YAGNI as an excuse to ignore future architectural requirements. It is an iterative dance between Intentional Architecture and Emergent Design.

KIS: KEEP IT SIMPLE

There is a quote that is attributed to Einstein: "Everything should be as simple as possible but not simpler". We should keep that in mind as we design and write our code. We should build the simplest thing possible that meets our requirements. Again, we should not use KIS as an excuse to ignore future architectural requirements.

ALWAYS BE HUMBLE AND CONTINUE LEARNING

We should always strive to be the best developer we can be but know we will never be *that* good. We will never be good enough to ignore others. We will never be good enough to not make a mistake. We will never be good enough to not need to continue learning. Technology changes every day and so should we.

This TIP was written by Lorenzo De Leon, who specializes in Financial Services and Healthcare & Life Sciences. Lorenzo welcomes comments and discussion on this topic and can be reached at Lorenzo.deleon@trexin.com.