**97 things every programmer should know**

**▼ Title Of Book and Chapter**

**Chapter 9 Check your Code first before Looking to Blame Other**

**▼ What are Three Things I learned today**

**1. Question your own assumptions and the assumptions of others**

Before – Question your ability if you really know how to do things or if you are just saying things easily that I can do that even if you’re not really sure that you can.

After - Each one of us has different assumption. So for example I can say that I know Paul can do Back End. But Paul himself know that he can’t. It’s just my assumption for him.

**2. Sherlock Holmes' advice, "Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth,"**

Before – If you know the truth no matter how impossible it is even if you are accused that you made a mistake but you know in yourself that you are innocent. It's okay because the truth will prevail. So you shouldn't blame others because you don't know the consequences when you are accused of no fault.

After - When the code comes from you and you encounter bugs you have to accept it and don't blame others because you are the one who made it and you know yourself that it is you. And before you blame others check your code first

Before – Checking your own code before blaming others because you don’t know how it feels in it comes back to you.

After – Because it’s hard for the developers to believe that their code is broken it is because maybe they really believe that it is a 100% sure that it’s running but then how come it is broken. Even it’s hard to believe but you also have to accept that developers are lacking something so don’t blame the compiler for it. And accept that we are not perfect.

**▼ Title Of Book and Chapter**

**Chapter 10 Choose Your Tools with Care**

**▼ What are Three Things I learned today**

Before – Making sure you choose the tools correctly and handle it with care

After - If you care about the tools that you choose then it has a big possibility that in the future you can still you use it.

Before – So if example if you choose a good employee and you care for them then they will not abandon you, if you treat them correctly.

After - So referential transparency is a fancy term that is attached to a computable expression.

Before – Choosing a tool is very hard because you to consider a lot of things.

After - But choosing a tool needs a lot of knowledge and time. Knowledge of knowing the functions of that tool and it takes time taking care of the tools that you choose.

**▼ Title Of Book and Chapter**

**Chapter 11 Code in the Language of the Domain**

**▼ What are Three Things I learned today**

Before – I don’t have any idea about this.

After - As what I understand this code in a language of the domain is that you need to be specific.

Before – I don’t know if my understanding after reading this code in the language of the domain is correct.

After - Making a domain concepts explicit in your code is to gather the intent of the other programmers code and for them to understand it easily

Before – Before I don’t have any idea of this domain.

After - Domain is a specific used for a specific purpose.

**▼ Title Of Book and Chapter**

**Chapter 12 Code Is Design**

**▼ What are Three Things I learned today**

**1.** **Our ability to predict time lines will fade away**

Before – So I predicted that tomorrow will rain but then tomorrow there’s no rain.

After - It is because for example we are given a time to do a design and we need to finish it before end of February and didn’t make it in the predicted day of finishing the design because the predicted time is not enough for us to do our design. So the reliable time slip away.

**2. The pressures of a competitive economy still apply**

Before – The pressure of making a design on time.

After - The pressure is on the employee that the boss wants the design to do it fast so that the boss can deploy it early and a lot of consumer wants the design because that is new to their eyes so it will look good to them.

**3. Design crisis**

Before –Design crisis is the incomplete design.

After - So design crisis is the quality of the design but the pressure of using the incomplete design.

**▼ Title Of Book and Chapter**

**Chapter 13 Code Layout Matters**

**▼ What are Three Things I learned today**

**1.** **Easy to scan**

Before – There are some people are really good at visualizing.

After - If you’re doing some coding make sure that it is arrange and it is easy to scan because a lot of people are good at visualizing like they just see your codes or design but they already know pattern or they have the idea.

**2. Expressive layout**

Before – it is nice to see that your code align because it’s clean.

After - The layout of the code is also part of this expressiveness. And it is more important to me that the line breaks and groupings represent the intent of the code rather than just the syntax of the language.

**3. Compact format**

Before –Saving space if you do this compact format.

After - If you do this compact format you make your code less and..