**National University of Computer and Emerging Sciences**

# Assignment # 3


**Student:**

Abdul Rehman …… 19L-1135


**Section:**

Professional Practices in IT (BCS-7C)

**Instructor:**

**Waqas Manzoor**

# Part#1
# Professional Software Development

## Section#1

They are as given below

1. Rule of Clarity:
   As maintenance of the code is carried out not by the computers but by the developers and their staff, so we must make sure that the code is written in a way that is as clear to the junior developers as it is to the main developers and senior developers, cause doing so saves time and provides more efficiency in development. E.g.

   Consider you have a code and you have to introduce some new functionality in a few days. You have been assigned some junior developers to help with your task, but they have not been able to clearly understand the code you have developed. If they have to spend a lot of time first understanding your code to get clarity of its functionality, it will eventually lead to slowing down your work speed and unnecessary waste of time, clarity of code is very important.

2. Rule of Separation:
   By introducing separation, it is possible to introduce new changes more easily without breaking the main mechanisms of the code. It allows for flexibility and effective development. E.g.

   Consider you are developing a game and you developed both the player profile and the weapon profile in the same module. Suddenly, while developing, you have to introduce new functionality to the weapon profile, but due to developing both profiles in the same module, its rigidity demands that a change in the weapon profile causes a change in the player profile. This will not only cause more time cost in development but also restrict the developer to free test multiple test cases on the weapon profile. So, separation not only provides flexibility in code but also provides effective testing.

3. Rule of Simplicity:
   Programmers are bright people who are (often justly) proud of their ability to handle complexity and juggle abstractions but complexity in code leads to more work in introducing change or error handling. Therefore, it is best to just “Keep it Simple”. E.g.

   If a developer is achieving a simple functionality in 2000 lines of code, which could easily be achieved in 100 lines using built-in libraries, it will only lead to difficulty in event of

code failure as it will be hard to identify the problem and it will take longer to debug such large amounts of code without spending a considerable amount of time on it.

## Section#2

They are as given below

1. Take Responsibility:
   If something doesn't go according to plan, instead of blaming others for it, you must understand where you did wrong, take responsibility, and improve from your mistakes. If an issue occurs, instead of making excuses, think about options to tackle that problem. E.g.

   Consider if you have a deal with a vendor and he suddenly back out, you must have a contingency plan on how to reuse or recycle the work you have done, such as reusing the already written code in any other program or project that you are working on.

2. Know when to Stop:
   Sometimes, we programmers get a little obsessed with our work, we don't know when to stop and we end up spending too much time on one module and end up making errors in the rest of the project leading to a bad product. Don't spoil a perfectly good program by over-embellishment and over-refinement. Move on, and let your code stand in its own right for a while. It may not be perfect but sometimes "Perfection leads to Disaster". E.g.

   Consider that you are developing a program, you have made it but in the testing phase, a module of your program was giving results that were under the acceptance criteria but weren't the optimal results that you desired, so you spend a little more time of it rather than testing the rest of the modules of your code. This can cause a potential for an error in the rest of the modules.

3. Your knowledge Portfolio:
   We think that programmers know it all and have knowledge about every fact related to their domain but it's not always the case. Managing your knowledge Portfolio is like managing your Financial Portfolio. You should not invest in multiple places if you can't handle it, furthermore, you should keep a regular investment in the places that are very necessary. E.g.

   If you are a Java developer, then you should have some knowledge in Python as well but you shouldn't be overinvested in learning and maintaining the python skills that you forget to update your knowledge on the latest development in the Java Domain. Always have good knowledge in the field most related to you as the saying goes "Jack of all trades but master of none."

# Part#2

## Topic: Playing Well with Others (ch#6)
## Book: Peopleware, 3rd Edition

## Summary:

After the globalization of virtually everything, assembling a team of people coming from various cultures, mindsets and counties look more like a United Nations task force. It can be difficult to combine the various elements into a team, but there are also advantages, some of which are discussed below:

**First, the Benefits:**
We can all recall a time when technology teams typically lacked women's participation or were almost entirely deprived of women. In Software Engineering Economics, the industry increased from a minimum of zero billion dollars per year to thirty billion dollars per year in just over 25 years. The majority of that money was spent on people. And from where did all of those people come? Because those industries were also expanding, they couldn't come from the same sources that supplied the other high-tech industries. Because there weren't enough male math and computer science majors coming out of the university systems at that time, they couldn't be the other obvious choice. The trick was to use the untapped resource of educated women whose previous career options were severely restricted. Since few universities offered useful software skills courses to their students, the hiring companies had to teach women specific skills like programming, debugging, and systems design.

“More than just hours worked, women brought much more to their new industry.”

They altered team members' interactions and organizational structures. They added fresh examples from ballet, child-rearing, and family dynamics to our tired sports analogies. They introduced new styles, which are referred to as "seat of the skirt management," as they entered management. Today, an all-male team appears to be lacking in strength and enthusiasm, but the addition of women makes a significant impact on that department.

**Food Magic:**
The dissolution of national boundaries brought new varieties of diversity, additional nationalities, and brand-new cultural patterns to our projects as a result of common markets and globalization. Even though adjusting to these changes might have been difficult, they also brought wealth. For a simple example, consider cuisine, although foods like Chinese food, Italian food, and Asian food are now commonplace in the modern world, our ancestors probably never saw one.

So, by combining different ingredients, we were able to make new and delicious dishes in the tech industry.

**True, but...**
However, a team's capacity to absorb new information has its limits. We are human beings after all. You have just had to integrate new people into your project, you need additional intent to try not to lease people like you lease vehicles.

Team success takes time, and the team's composition cannot be changing for the majority of that time. Your team probably won't work well if you have to resort to contract labor as a reactive strategy. In fact, the workforce you oversee almost certainly will not be a team. So, knowing how to combine the right people together to form an ideal team and knowing how they will get along with one another greatly affects the success rate of a project.