CHAPTER 2

PROFESSIONAL ETHICS AND ISSUES IN SOFTWARE DEVELOPMENT

2.1 INTRODUCTION

A Software professional is always responsible for successful development of software. Identifying human resources is more difficult than identifying the software or hardware assets. Concerned people with sufficient knowledge and experience should be assigned to a particular task. Engineers and scientists share a basic drive to accomplish what they want to so as to establish their own unique identity to have a rich in the profession. It is therefore necessary to match profession with their work assignment. Enhanced technical issues and impressive interpersonal skills are very much involved in it. The development teams face many challenges and hardships as they are pressurized by several factors during the software development phase.

Professional skill and attitude differ from person to person. It is therefore necessary to identify the talent and attitude of an individual that can be appropriately applied to the different phases of the software development (Orit Hazzan and Jim Tomayko 2005). The best result can then be obtained only if different tasks are allotted among different software professionals according to their nature and ability. Hence it is imperative to identify the unique skill of an individual and apply the same to the software development process, thus reducing the level of risks.
2.2 IDENTIFICATION OF TECHNICAL PROFESSIONALS FOR SOFTWARE DEVELOPMENT

Every field of specialization has a unique set of talents that is responsible for its success. The Professional can generally succeed as long as the talents are reasonably consistent with the requirement of the work. The talent pool is the most important asset of an organizazation. Creative ideas originate from it and solve the key problems to produce the most successful product.

2.3 LINE OF CODE ANALYSIS

This analysis is intended to identify risk in the perspective of a programmer’s knowledge. This is done through the comparative study on the programming skills of different programmers in a particular scenario as shown in Table 2.1. Every developer has his own technique to develop a program. The style of program writing of a programmer is entirely different from others. Hence this study promotes an idea of choosing the best programmer among a crew of programmers based on three levels of program writing. They are:

1. Low level program - Low level program involves simple programming with limited number of variables. Example: simple mathematical calculations.

2. Medium level program - Medium level Program consists of variables at multilevel systems. Example: Use of Inheritance to access the variable from base class.

3. High level program - High level Program involves complex data variables in the system which depicts the importance of
the variable and its scope within the system. Example: Functions used for online quiz system.

The ‘level’ reveals the complexity levels of the program and not the types of software.

The calculation metrics such as Line of Code (LOC), Program Volume, Compilation Detail and Error are used for analysing the skill levels of a program. They are detailed as below.

**LOC** (N): LOC represents Line of Code. LOC is the total number of lines present in a program. If the LOC is large, then it is generally assumed that the complexity of the program is also large.

**PROGRAM VOLUME** (V): Program volume can be represented by an Equation(2.1).

\[
V = N \log n
\]  
(2.1)

where,

N - Total number of lines present in a program

n - Sum of the total number of operators and operands present in a program.

If program volume is larger, it seems to have much complexity because there must exist many operators and operands.

**COMPILATION DETAILS:** Compilation details are based on the program errors. This error is divided and analyzed into various categories.
**ERROR:** Error is defined as an incorrect output or incorrect action. Three types of Error exist. They are

1) Known Errors: Errors about which the programmers know and are able to overcome. Faulty Syntaxes are such error.

2) Predictable Errors: These errors are those which the programmers do not know the exact solution. But they try out some solutions. For example: logical errors.

3) Unpredictable Error: The programmers may not be able to find a solution for a particular error. Such errors are known to be unpredictable. For example: Some linker errors or a machine error that may be implicit or beyond the recognition of the programmer.

Table 2.1 shows the analysis of a program based on different skill levels of programming. It is a sample taken from an organization. The data are collected from various analyses conducted in an organization using different programs.
Table 2.1 Error Analysis of Different Programs from different Skill levels of Programmers

<table>
<thead>
<tr>
<th>Programmer</th>
<th>Line of Code(LOC)</th>
<th>Program Volume</th>
<th>Compiler details</th>
<th>No. of known Errors</th>
<th>No. of known Errors Cleared</th>
<th>No. of known Errors not Cleared</th>
<th>No. of predictable Errors</th>
<th>No. of predictable Errors cleared</th>
<th>No. of predictable Errors not cleared</th>
<th>No. of unpredictable Errors</th>
<th>No. of unpredictable Errors cleared</th>
<th>No. of unpredictable Errors not cleared</th>
<th>Total Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low level Program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>24</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>25</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>25</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>27</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Medium level Program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td>83</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>54</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>99</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>83</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>56</td>
<td>93</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>High level Program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>123</td>
<td>194</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
<td>189</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>125</td>
<td>198</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>123</td>
<td>194</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>129</td>
<td>204</td>
<td>10</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Figure 2.1 Graphical Representation of Programmers Vs Errors

The bar chart shown in Figure 2.1 presents the results of a comparative study for programming skills of different programmers in different aspects of human ethics of a coding development. It also depicts the comparison of various capabilities of the programmers who are working in different platforms. This shows that the capability level of programmer differs from one another.

2.4 DICTORIAL RISK FACTOR FOR SOFTWARE PROJECT CODING

Risk attack in software projects: The highest risk attack starts from the analysis phase and goes through requirement design, implementation and ends in the testing phase.

Figure 2.2 shows the areas of a project where the project manager has high level of risks in the implementation coding phase. As long as the risks are determined and securely mitigated, the quality of risk will slowly move on to its predicted path. If risks are not identified and not securely mitigated, the bugs will rapidly increase in the implementation phase and finally the software project might fail to satisfy the requirements of the customer.
Risk due to inadequate knowledge: Even though a software developing team could identify the risks from design phase, it has to overcome many risks that occur at coding phase also. Professionals, who lack in knowledge in a particular area, tend to increase risk in the product. The primary goal of a software development team is then to develop a code and document the development that should meet the project’s requirement. The primary issue is that the software must be maintainable and reusable. According to the impact level of risk occurrence in a system, it is classified into three levels of risk identification. They are

1. Low Level
2. Medium Level
3. High Level

The example for these risk levels given in Table 2.3. show risk controlling factors that depicts the various factors which influence the risk in a system. The risk level in percentage is determined by using the following Equation (2.2).
Percentage of Risk Level Identified in the system = \frac{\text{Number of Risk Factors which produce major effect in the system}}{\text{Total No. of Risk Factors in the system}} \times 100 \quad (2.2)

By using the above equation, the percentage of risk level is found as presented in the following Table 2.2.

**Table 2.2 Different Percentage of Risk Levels**

<table>
<thead>
<tr>
<th>Risk Level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>0 % to 30 %</td>
</tr>
<tr>
<td>Medium</td>
<td>31% to 60 %</td>
</tr>
<tr>
<td>High</td>
<td>61 % to 100 %</td>
</tr>
</tbody>
</table>

**Table 2.3 Example for Risk Controlling Factors**

<table>
<thead>
<tr>
<th>Factors</th>
<th>Example</th>
<th>Risk Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>Locate Error</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Analyze error</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Estimate error</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Data relation error</td>
<td>Low</td>
</tr>
<tr>
<td>Bugs</td>
<td>Control bugs</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Runtime bugs</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Events of human error</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Software attacks</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Unauthorized access</td>
<td>High</td>
</tr>
<tr>
<td>Faults</td>
<td>Initialization problems</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Incorrect use of Defaults</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Wrong Data type, wrong measuring unit</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Wrong Boundary value</td>
<td>Medium</td>
</tr>
<tr>
<td>Failures</td>
<td>Transient Failure</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Recoverable</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Unrecoverable</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Non corrupting</td>
<td>Low</td>
</tr>
</tbody>
</table>
Components that invoke bugs; problem caused by different parameters, Divide checks are some examples for the various factors which affect the implementation. Coding in software projects cause risk due to inadequate knowledge of the programmers.

They are briefed below:

**Component invocation bugs:** Any wrong component typically a faulty one or a non-existing component is invoked.

**Example Program written in C++**

```cpp
class count {
public static void main (string args[ ])
{
...
for (i=0;i<10;i++)
{
 int count ;
count ++;
}
System.out.println (count);
}
```

This program shows an error: **unable to find symbol count.** The error has occurred because the scope of count is only inside the inner loop, and the component is invoked at the wrong place.

**Problem with parameters:** Parameters may be incorrect due to their number order, data type, value, connections etc.
Example Program written in C++

```cpp
class rectangle
{
...
rectangle (int x, int y)
{
    i =x;
    w = y;
}
}
class area
{
    public static void main (arg[])
    {
        rectangle rect1 = new rectangle (15);
    ...
    }
}
```

This program shows an error: **Mismatch in number of arguments.**

In this program a two argument constructor is defined by a single argument constructor. Hence type mismatch in number of arguments passed occurs.

**Divided by zero:** In any arithmetic expression, if the denominator is 0, then an error occurs.

Example program written in C++

```cpp
class test
{
public static void main(string args[])
{
    int a = 5;
    int b = 2;
    int c = 3;
    a = a / ( c-b-1);
}
```
This program shows an error: The expression result in a denominator of zero and hence an error occurs.

Software projects that are based on five phases of software development life cycle may either be language oriented or application oriented. It is developed by programmers. If programmers have inadequate knowledge, then the risk factors increase in the form of errors, bugs, faults and failure. The general structure of risk is shown in Figure 2.3.

![Figure 2.3 General Structure of Risk](image)

### 2.5 TWO LEVELS OF SOFTWARE PROFESSIONALS

Software comprising of several modules are generally developed by a team of professionals in a software company. Each module is developed by
an individual or a group of Professionals. It is well known that a risk in the software depends on various factors such as skill requirements, scheduling and cost. Among these factors, skill of an individual of a particular language in which the product has to be developed will give a greater impact on risk in that software. In general, Software professionals are of two types namely A) Vertical Software Professional (VSP) and B) Horizontal Software Professional (HSP) as shown in Figure 2.4. VSPs are those who are experts in a particular language and they may not be well-versed in other languages. HSPs are those who are not experts in a particular language but they are moderately fair in many languages. For example, Professionals of group ‘A’ are experts in “C” language and capable of solving any problem in “C” language. Professionals of group ‘B’ are not experts in all software languages but they just know many languages. They are likely to make mistakes in the developing cycle of the software.

![Figure 2.4 VSP vs HSP](image-url)
2.6 ALLOCATION OF SOFTWARE PROFESSIONALS USING SET THEORY

A fundamental concept of set theory is that a membership belonging to a set or any object enables to a member or an element of that set (Trembley and Manohar 1989). The object in sets may be anything, say numbers, people, rivers, cars, or mountains. If an object x is a member of a set A, and represented x ∈ A, which may be understood as “x belongs to A” in other word “x is an element of the set A”, In contrast when x ∉ A; that is “x does not belong to A”, when an object x is not a member of the set ‘A’ (Tremblay and Manohar 1989). Similarly when VSP belongs to team of HSP or vice-versa of a project. Thus the notation of membership of an element in a set is understood. Another basic concept in set theory is Inclusion. Let A and B be any two sets. If every element of A is in element B, then A is called the subset of B, or A is said to be included in B, or B includes A. Symbolically, this relation is denoted by A ⊆ B, or equivalently B ⊇ A. Alternatively, A ⊆ B ⇔ (x) (x ∈ A → x ∈ B) ⇐ B ⊇ A.

Example  A = { 1,2,3} B={ 1,2} C= (1,3} and D = {3} then B ⊆ A , C ⊆ A , D ⊆ A or {1,2} {1,2,3} {1,3} {1,2,3} {3} {1,2,3}. This example denotes all types of software people constitute a mixture in implementation of a coding. In other words, all the probability of VSP and HSP are mixed in coding development for a project.

The mathematical abstraction given below will show, how the risk reduction is possible in developing a software. Let set ‘A’ be of VSP and let set ‘B’ be of HSP. Both the sets A and B have N number of Professional in each A = { A1,A2,A3,........N} and B = { B1,B2,B3,........N}. 
By using Venn diagram of inclusion set theory i.e.,\( n(A \cup B) = n(A) + n(B) - n(A \cap B) \). Where, \( n(A \cup B) \) refers to the strength or risk free code developed from union of VSP and HSP. Then \( n(A) \) refers to total skill of VSP, and \( n(B) \) refers to total skill of HSP. \( n(A \cap B) \) refers to risk reduced by the combination of the groups when they are employed together in a project as shown in Figure 2.5.

![Venn Diagram](image)

**Figure 2.5  Risk Level Reductions by Combining VSP and HSP**

When both the groups of VSP and HSP are combined together for a project, then chain of risks can considerably be reduced. If sets \( A_1 \) and \( B_1 \) are disjointed, then the risk level will be very high. If element of \( A_1 \) and \( B_1 \) are intersected up to \( A_N \) and \( B_N \) as given in series(2.3), then the risk level can be considerably reduced by using inclusion set theory of Risk reduction chain of VSP and HSP as shown in Figure 2.6.

\[
(A_1 \cap B_1) + (B_1 \cap A_2) + (A_2 \cap B_2) + (B_2 \cap A_3) + (A_3 \cap B_3) + \ldots + (A_N \cap B_N)
\]  

(2.3)
In Figure 2.7 the Inclusion and Equality Set of VSP and HSP is shown. The ‘Z’ refers to risk reduction chain. If A,B,C are any three sets, then \( n(A\cup B\cup C) = n(A) + n(B) + n(C) - n(A\cap B) - n(B\cap C) - n(A\cap C) + n(A\cap B\cap C) \).

2.7 APPLICATION OF THE PROPERTIES OF SET THEORY FOR VSP AND HSP

The levels of risks, whether high or low, will depend upon properties of inclusion. The property of inclusion is represented by set theory. Let us assign two representations namely: \( A = \text{VSP} \) and \( B = \text{HSP} \).

Certain mathematical notations for these representations are provided (Table 2.4). These notations that use set operations have been
derived for these properties. The suggested notations have been validated (Tremblay and Manohar 1989). Depending upon a particular nature of properties, an appropriate mathematical notation of set theory is suggested (Table 2.4).

The level of risk would be low, if the properties satisfy the mathematical notation for the corresponding property. It they don’t, then the risk would be high. This is demonstrated in Table2.4.

**Table 2.4 Usage of Set Theory in Assigning Software Professionals in Software Development Projects**

<table>
<thead>
<tr>
<th>S.No</th>
<th>Property</th>
<th>Syntax</th>
<th>Explanation</th>
<th>Level of risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Inclusion of Sets</td>
<td>A ⊆ B ↔ (x) (x ∈ A → x ∈ B) ↔ B ⊆ A</td>
<td>VSP is included in HSP, or HSP is included in VSP.</td>
<td>Low</td>
</tr>
<tr>
<td>2.</td>
<td>Equality of Sets</td>
<td>A = B ↔ (A ⊆ B ∧ B ⊆ A)</td>
<td>VSP and HSP are equal in number.</td>
<td>Low</td>
</tr>
<tr>
<td>3.</td>
<td>Intersection of Sets</td>
<td>A ∩ B = {x</td>
<td>(x ∈ A) ∧ (x ∈ B)}</td>
<td>The intersection of any two sets of VSP and HSP, written as VSP ∩ HSP is the set consisting of professionals who are of the same category of skills both in VSP and HSP.</td>
</tr>
<tr>
<td>4.</td>
<td>Disjointed Sets</td>
<td>A ∩ B = ⌀</td>
<td>VSP and HSP have no talents in common</td>
<td>High</td>
</tr>
</tbody>
</table>

**2.8 RESULTS AND DISCUSSION**

Line of code analytical results are used in understanding the various skills of programmers in code development. It also represents the growth of system with various levels of programmers involved in the development of a
system. The following graph shown in Figure 2.8 depicts the efficiency of risk levels when used with different people (HSP and VSP) for development. It shows that the risk level gets reduced when both the HSP and VSP are combined. It satisfies the set theory relationship based on the HSP and VSP personnel behaviors.

![Personnel Vs Risk](image)

**Figure 2.8 Personnel vs Risk**

The risk level of the system can be reduced considerably when the HSP and VSP professionals are together. It also predicts the goal of a system when they comprise both the professionals for effective achievement of the goal. In order to make a system efficient, the reusable components need to be used. The reduction of risk then would be effective during development time, on effective memory utilization and on-time delivery of the products.