Assimilation of Software Process Management in Software Engineering Undergraduate Curriculum

Abdul Kadir Khan

College of Computing and Informatics, Haramaya University Ethiopia

qadirforu@gmail.com

Abstract – **In this paper, a case study of Haramaya University (Ethiopia) is discussed with the help of Watts Humphrey's Personal Software Process throughout the software process management course being taught in 2nd year (2nd semester) Software Engineering undergraduate curriculum. It includes a description of the assignment's objectives, a discussion of the activities engaged in by students, an explanation of how the activities were amalgamated into the curriculum, a detail study of the data gathered and an evaluation of the impact and importance of the assignment. Finally ideas about amendment and expansion of the assignment's concepts are offered.**

Keywords: **Personal Software Process, Process Management, Software Engineering, Undergraduate Curriculum.**

1. INTRODUCTION

Process is a term used to describe the people, methods, and tools used to produce software products. Improving the quality of the product is believed to be based on improving the process used to develop the product. Software engineering process is defined as the system of all tasks and the supporting tools, standards, methods, and practices involved in the production and evolution of a software product throughout the software life cycle. Process management supports improvement of the defined process through measurement and feedback. Current implementations of process management combine three steps: definition, control, and improvement [1].

The existence of a gap between industry software engineering needs and academic software engineering education has been widely reported [2,3,4]. Industry requires software engineers but Universities are educating computer scientists [3]. A recent Software Engineering Institute report succinctly states the problem: "Students are not prepared to make the jump between computer science and software engineering in school to software engineering in the professional world." One reason for the gap between what students learn in Universities and the skills required by the software industry is that few existing academic programs address the competencies expected of incoming software engineers. These are programming in the large, ability to work in teams, knowledge of process, quality and the ability to estimate time and labour costs. Computer science education, where most software engineering education occurs, too often focuses on individual contributions rather than managed group efforts involving these competencies. Group efforts, however, are the norm in the software industry [4].

Current curricula do not allow students to develop real software, but rather focus on what can be called "toy projects in toy situations" [2]. One approach to overcome this has been the introduction of project courses where students are grouped into teams and assigned a "realistic" (but toy-sized by industry standards) problem to solve. This has not proven to be effective for several reasons:

1) In Ethiopia, software industry area is very limited.
2) Resources are limited.
3) Most of the academic staffs of Haramaya University lack knowledge of practical aspects of software development and project management.
4) Since the university faculty members are not aware of industries primary issues, many topics critical to software companies are not even discussed.
5) Students are expected to apply methods and techniques of software development to a "realistic" problem while they are learning these topics.
6) Very little instruction on planning, teamwork, and intergroup coordination is given.
7) An academic semester is too short to learn and practice all the life cycle operations.

It is not possible to design a project/program that will address all of the above issues. However, by amendment of the existing software engineering programs, it will be able to get closer to graduating students with qualifications that are needed currently in industry. The College of Computing and Informatics faculty at Haramaya University have developed a software engineering program that addresses some of the issues. Some of the key elements of the program are:

1) Software engineering concepts introduced into the first year courses.

2) Introductory concepts were expended on in subsequent years.
3) Software development practice is introduced and emphasized through at least ten courses.

In the following parts of this paper, a detailed research has been used to employ the inclusion of software engineering practices across the curriculum.

2. IPQ/PSP RESEARCH

The **Personal Software Process** (PSP) is a structured software development process that is intended to help software engineers understand and improve their performance, by using a "disciplined, data-driven procedure". The PSP was created by Watts Humphrey to apply the underlying principles of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) to the software development practices of a single developer. The PSP is similar to the Capability Maturity Model (CMM), except that it focuses on the personal process. It claims to give software engineers the process skills necessary to work on a Team Software Process (TSP) team. The PSP concentrates on the work practices of the individual engineers. The principle behind the PSP is to produce quality software systems; every engineer who works on the system must do quality work. This means that almost everyone associated with software development must know how to do discipline engineering work. When engineers use the disciplined approach to software development included by PSP, they learn to become more competent engineers producing quality software products on schedule [5-12].

The PSP is designed to help software professionals consistently use sound engineering practices. It shows them how to plan and track their work, use a defined and measured process, establish measurable goals, and track performance against these goals. The PSP shows engineers how to manage quality from the beginning of the job, how to analyze the results of each job, and how to use the results to improve the process for the next project.

1.1 The Principles of the PSP

Right way and the right approach is to be followed to do a software engineering job, engineers must plan their work before committing to or starting on a job, and they must use a defined process to plan the work. To understand their personal performance, they must measure the time that they spend on each job step, the defects that they inject and remove, and the sizes of the products they produce. To consistently produce quality products, engineers must plan, measure, and track product quality, and they must focus on quality from the beginning of a job. Finally, they must analyze the results of each job and use these findings to improve their personal processes. Software design is characterized as a set of practices and implementation techniques that allow the construction of marketable software systems that provide form and function satisfying to users [5].

The PSP design is based on the following planning and quality principles:

- Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on their own personal data.
- To consistently improve their performance, engineers must personally use well-defined and measured processes.
- To produce quality products, engineers must feel personally responsible for the quality of their products. Superior products are not produced by mistake; engineers must strive to do quality work.
- It costs less to find and fix defects earlier in a process than later.
- It is more efficient to prevent defects than to find and fix them.
- The right way is always the fastest and cheapest way to do a job.

In the 2011-2012 academic years, software process concepts and activities were introduced into Software Process Management course. An early version of Watts Humphrey's *Introduction to the Personal Software Process* [5] was used to teach the course contents. The book is written for students new to computer science and software engineering. Its narrative style, examples and exercises have been customized to 2nd year software engineering students. There is a major stress on time management, and the PSP objects on estimation and design has been made easy. Also, there are no prearranged programming assignments for the PSP activities, so the objects can be used with a variety of different programming assignment profiles.

This assignment has been called "Improving Product Quality" (IPQ). (All through this paper, the reference to this assignment will be as IPQ/PSP.) Main goal of this research was to emphasize to the students, from the very beginning, how important quality was in their work. More particularly our objectives were:

- Provide students with learning experiences that will help them to understand the importance of a disciplined approach in the development of software;
- Engage students in activities in which they use the elements of a defined personal software process;
- Provide a foundation for further development/ improvement of the student's personal software process and its use in individual and team projects.

The IPQ/PSP research was divided into two parts:
1) Time Management Activity;
2) Quality Improvement Activity;

Although all students entering our Software Engineering 2nd year (2nd Semester) courses had programming experience, they were yet not ready for a formally defined software development process so in the beginning of the semester, students could best assimilate and assistance from the time management activities. After 1 month, PSP activities were formally introduced. Students carried out project planning for each project. This included using historical data to estimate size, effort and quality (defect projection). Actual data for each project was collected and recorded on a summary sheet. At this level the students (after finishing 1 month and their first year in college) had the maturity and were ready for a more disciplined way to develop their programs. By delaying the introduction of PSP for a further semester (or year) would allow sloppy and undisciplined programming practices to become more entrenched and make such practices much more difficult to change.

1.2 Time Management Activity

During the first offering, students were asked to keep an accurate time log which represented the time they spent on software engineering 2nd year (2nd semester) tasks. This time included interruptions down to the minute. The IPQ/PSP material was distributed rather uniformly throughout the semester. DQW assignments consisted of time management exercises where students were asked to keep track of the time they spent on the following four activities:

- ATTENDING CLASSES- the time spent in class
- IPQ- the time spent on filling out forms and performing necessary calculations.
- PROGRAMMING- the time spent on a programming assignment
- READING- the time spent preparing for class, including class, quiz and exam preparation

There were a total of four weeks of IPQ/PSP assignments. Each weekly assignment consisted of two sub assignments: a planned/budget time that a student would spend on any of the five activities during a week, and the actual time the student spent on each of the five activities. Table 1 – 4 shows the average budget error (for 30 students) for the CLASS, READING, PROGRAMMING and IPQ tasks for weeks 1 through 4. The budget error is computed using the below formula:

Budget error = 100 * (actual - budget) / budget;

TABLE 1
1ST WEEK BUDGET PLAN

Activity	Budget Minutes	Actual Minutes	Budget Error
Attending Classes	900	840	-6.666666667
Reading	600	800	33.33333333
Programming	600	1000	66.66666667
IPQ	420	400	-4.761904762

TABLE 2
2ND WEEK BUDGET PLAN

Activity	Budget Minutes	Actual Minutes	Budget Error
Attending Classes	943	840	-10.92258749
Reading	720	912	26.66666667
Programming	900	1105	22.77777778
IPQ	410	395	-3.658536585

TABLE 3
3RD WEEK BUDGET PLAN

Activity	Budget Minutes	Actual Minutes	Budget Error
Attending Classes	900	800	-11.11111111
Reading	700	840	20
Programming	950	1158	21.89473684
IPQ	400	405	1.25

TABLE 4
4TH WEEK BUDGET PLAN

Activity	Budget Minutes	Actual Minutes	Budget Error
Attending Classes	900	650	-27.77777778
Reading	600	800	33.33333333
Programming	600	1000	66.66666667
IPQ	400	410	2.5

Research finding of time management activity through weekly budget plan is given below:
1) The error for PROGRAMMING and READING indicate more irregularity in this data than for the IPQ and CLASS data. This can be attributed to the deviation in assignment difficulty and course topics, and the fact that students are learning new objects for each assignment.
2) The CLASS and IPQ data budget show little error. This can be attributed to the fact that these tasks have less deviation from week to week and, hence, are easy to predict.
3) Notice that in the last week of ATTENDING

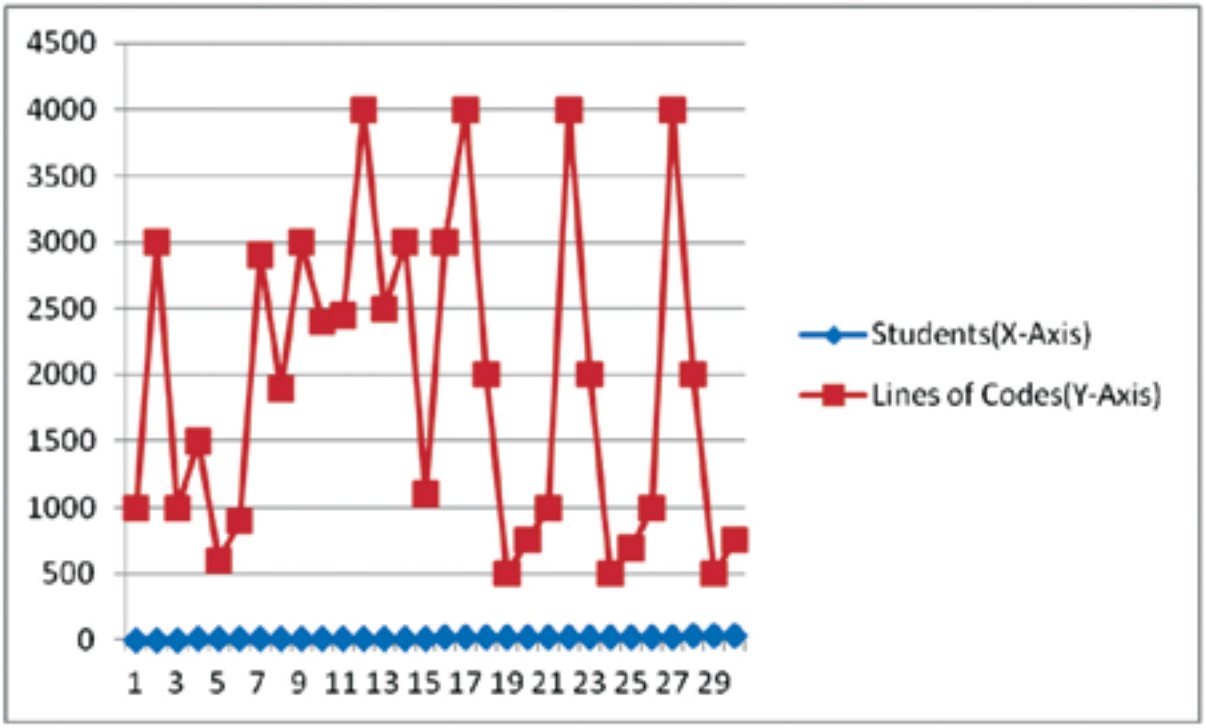
CLASSES data the error is relatively large (-27%). There was an annual college meeting (no class) that week and a close look at individual data showed the students were not aware of meeting when their budgets were made.

4) In 2nd and 3rd week budget plan, budget error for PROGRAMMING and READING is less compare to 1st week budget plan because after 1 week, students somehow got the better idea of PROGRAMMING and READING.
5) But again in 4th week budget plan, budget error for PROGRAMMING and READING is more compare to other week budget plan because now students are reading and writing advance codes which are more time consuming than they expected.
6) Budgeting IPQ minutes is almost equivalent to actual minutes.

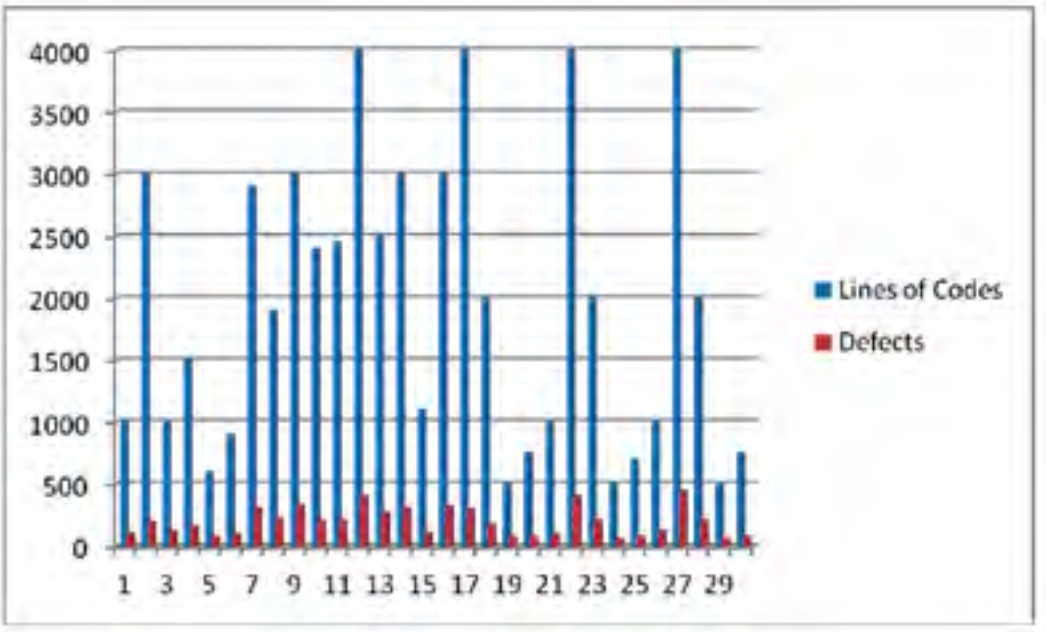
2.3 Quality Improvement Activity

This activity was introduced after one month. The quality improvement activities were concentrated on software defect management and used the second half of Humphrey's introductory text [5]. A simplified version of PSP was combined with the usual 2nd semester topics (introduction to data structures and algorithms) [3]. Students were asked to estimate and record data about effort in each of six phases of a program development process: planning, design, code, compile, test, and post-mortem. As the course progressed, students were asked to identify and record their defects. They cited the phases in which defects were injected and in which they were removed. A primary focus of this effort was to encourage thorough and productive code reviews.

As an example, consider Figure 1, and figure 2. Figure 1 shows the defects/LOC in first half (almost equivalent to 4 weeks) of the quality improvement phase for all students for different programs (lines of codes may vary from one student to another).



(a)

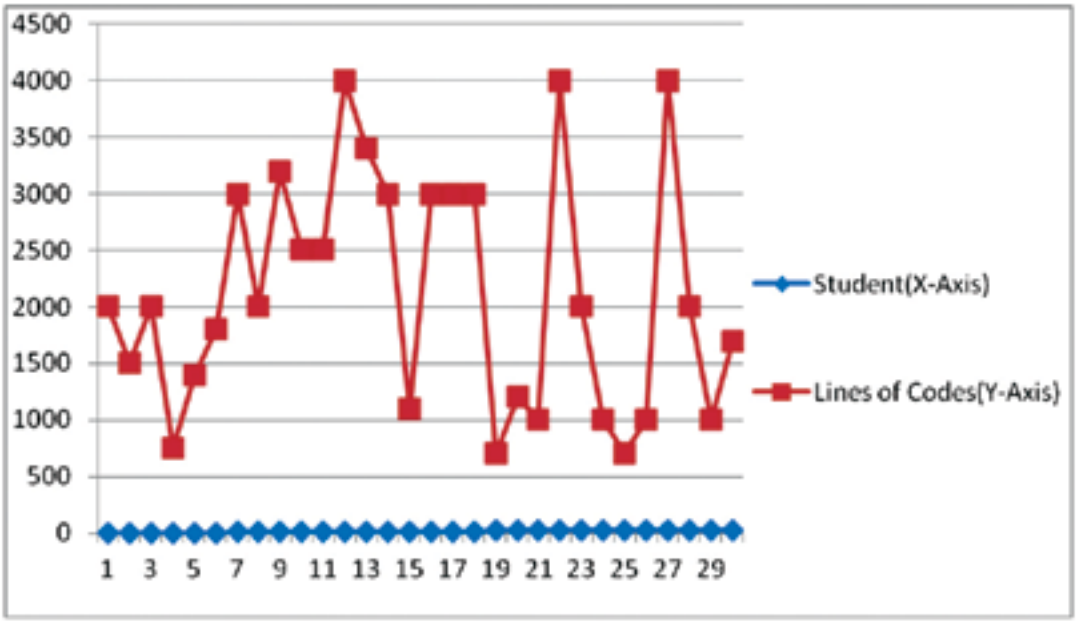


(b)

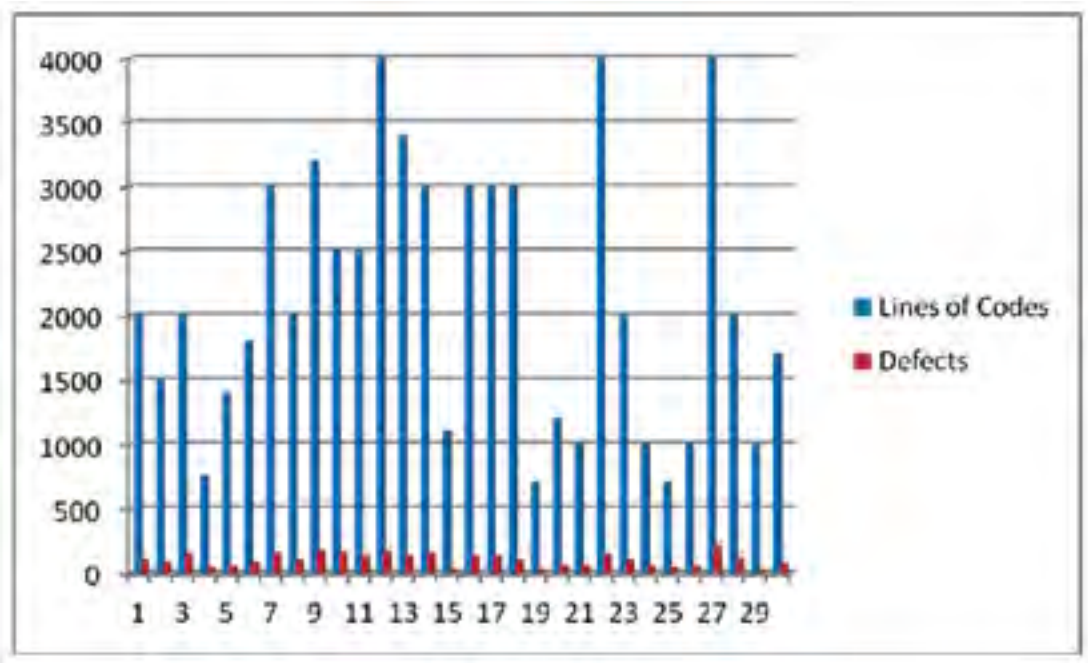
FIGURE 1: QUALITY IMPROVEMENT PHASE 1

Figure 2 shows the defects/LOC in second half (almost equivalent to 4 weeks) of the quality improvement phase for all students for different programs (lines of codes vary from one student to another).

For first half of quality improvement phase, most students had 100 or more defects/kLOC (kLOC = 1000 LOC), while for second half, most students had fewer than 50 defects/kLOC. Why such a drastic alteration? One explanation might be that formal code reviews began in second half of quality improvement phase and because of that the code review was so effective that program quality was dramatically improved. However, a closer look at the data reveals some other possible factors for this improvement; type of program or quality of testing.



(a)



(b)

FIGURE 2: QUALITY IMPROVEMENT PHASE II

3. IPQ/PSP RESEARCH FINDING

Based on the objective and subject evaluations of the IPQ/PSP activities, in the 2011-2012 academic years, the IPQ/PSP objectives have not been fully achieved. However, some meaningful progress was achieved. With additional efforts, and few changes in the approach can bring one closer to achieving the objectives.

Other research findings related to IPQ/PSP assignment are given below:

- Humphrey's introductory text [5] provides an excellent foundation and motivation for teaching IPQ/PSP concepts.
- There is a need for simplification and automation of forms, logs and record keeping.
- Teaching techniques must be modified as per the need.
- In time management activity, budget error decreases as plan moves from one week to another.
- Lack of commitment of some students towards their career goal.
- Quality improvement phases show that PSP plays a great role in improving the product quality.
- Students lack maturity level.
- The effort and effectiveness required for IPQ work versus apparent reward.

IPQ/PSP concepts need to be integrated throughout the Software Engineering curriculum.The author should simulate the usual appearance of papers in *SEIJ*.

4. CONCLUSION

The PSP is designed for use with any programming language or design methodology and it can be used for most aspects of software work, including writing requirements, running tests, defining processes, and repairing defects. When engineers use the PSP, the recommended process goal is to produce zero-defect products on schedule and within planned costs. It is not possible to design a project/program that will address all of the issues discussed in this paper. However, we believe that by amendment of the existing software engineering programs, we will be able to get closer to graduating students with qualifications that are needed currently in industry. IPQ/PSP data provides valuable information to teachers for analyzing student effectiveness in completing course work - especially programming projects. Time management activities play a great role to achieve the career goal. IPQ/PSP assignment develops an individual skill in writing quality software with few defects. Seeing through the benefits by implementing PSP in Software Process Management, it can further be beneficial for the other undergraduate curriculum (i.e. Information Technology, Computer Science, Information System etc).

REFERENCES

[1] W. Laurie, "Lecture Notes on Software Process Improvement," *IEEE Transactions on Software Engineering*, University of Texas at Austin, vol. 2, no. 1-1993.

[2] Dion, Raymond, "Process Improvement and the Corporate Balance Sheet," *IEEE Software*, July 1993.

[3] T. Hilburn, I. Hirmanpour, and A. Kornecki, "The Integration of Software Engineering into a Computer Science Curriculum," *Lecture Notes in Computer Science: Software Engineering Education Proceedings of the Eight SEI Conference on Software Engineering Education*, March 1995.

[4] A. .M Tucker, "Computing Curricula 1991," *Communications of the ACM*, vol. 34, no. 6, pp. 68-84, June 1991.

[5] Humphrey, S. Watts, "An Introduction to the Personal Software Process," *Addison-Wesley, Reading,MA,* 1997.

[6] A. K. Khan, "Amalgamation of Personal Software Process in Software Development Practice," *(Accepted to be Published) Star Journal,* April-June 2012.

[7] B. Boehm, "Software Engineering Economics," *Englewood Cliffs, NJ: Prentice-Hall,* 1981.

[8] M. Fagan,"Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal,* Vol. 15, No 3 1976.

[9] M. Fagan, "Advances in Software Inspections," *IEEE Transactions on Software Engineering,* vol. 2, no. 7, July 1986.

[10] W. Humphrey, "Managing the Software Process," *Reading, MA: Addison-Wesley,* 1989.

[11] W. Humphrey, "The Software Quality Index," *Software Quality Professional,* vol. 1,no. 1 December 1998

[12] M. Paulk, B. Curtis,& M.B. Chrissis, "Capability Maturity Model for Software," *Version 1.1 Pittsburgh, Pa.:Software Engineering Institute, Carnegie Mellon University,* 1995.

ABOUT THE AUTHOR

Abdul Kadir Khan. He is presently working as a Lecturer in Haramaya University, Ethiopia, having an experience of more than 5 years in teaching and from last 3 years, working in Ethiopia. He has completed his PG (Master of Computer Science and Applications) and graduation (B.Sc. (Hons)) both from AMU, Aligarh. From last 3 years, he is handling different software engineering courses (i.e. Software Process Management, Software Project Management, Fundamentals of Software Engineering etc) in Ethiopia. Currently he is working on Defect Analysis and Defect Prevention Research Project.