Volume : 3 | Issue : 1 | Jan 2014 • ISSN No 2277 - 8160

Research Paper | Computer Science

# The need for Software Engineering

**Prof. Sushmitha Sharad**

Faculty of Computer Science( M.C.A), Shree Narayana College of Commerce

KEYWORDS :

## Introduction

In the early days of computing the primary concern was with building the hardware or acquiring the hardware. Software was almost expected to take care of itself. The consent was that "hardware" is "hard" to change, while "software" is "soft," or easy to change. So, most people in the industry carefully planned hardware development but gave considerably less importance to the software. If the software didn't work, they believed, it would be easy enough to change it until it did work.

## Content

The cost of software amounted to such a small fraction of the cost of the hardware that no one considered it very important to manage its development. Everyone, however, saw the importance of producing programs that were efficient and ran fast because this saved time on the expensive hardware. People time was assumed to save machine time. However, as computing matured, programs became more difficult and complex. Projects grew larger whereas programs were written, operated, and maintained by the same person. Programs began to be developed by teams of programmers to meet someone else's expectations. Individual effort gave way to team effort. Communication and coordination which once were under one person had to occur between the heads of many persons. Thus the whole process was very complicated. As a result, communication, management, planning and documentation became critical.

As programs became more complex, the early methods used to make flowcharts were no longer satisfactory. In fact, without better methods of representation it became difficult for even one programmer to keep track of what he or she is doing. The times required to write programs and their costs began to exceed. It was not unusual for systems to cost more than twice and took weeks, months or years longer than expected to complete. Making a change within a complex program turned out to be very expensive. Often even to get the program to do something slightly different was so hard that at times it was easier to throw out the old program and start over. This, of course, was costly. Part of the evolution in the software engineering approach was learning to develop systems that are built well enough the first time so that simple changes can be made easily.

At the same time, hardware was growing ever less expensive. Tubes were replaced by transistors and transistors were replaced by integrated circuits. As the cost of hardware plummeted, software continued to be written by humans, whose wages were rising. The savings from productivity improvements in software development from the use of assemblers, compilers, and data base management systems did not proceed as rapidly as the savings in hardware costs. Some current developments, such as nonprocedural (fourth generation) languages and the use of artificial intelligence (fifth generation), show promise of increasing software development productivity, but we are only beginning to see their potential.

Another problem is that, once the program had been written, the client began to express dissatisfaction, and the producer, too, was unhappy. As time went by software developers learned to lay out with paper and pencil exactly what they intended to do before starting. Then they could review the plans with the client to see if they met the client's expectations. This was simpler and less expensive to make changes to this paper-and-pencil version than to make them after the system has been built. Using good planning makes it less likely that changes will have to be made once the program is finished.

## Conclusion

Take an example of a building construction. An architect can draw a floor plan. The client can usually gain some understanding of what the architect has planned and give feed back as to whether it is appropriate. Floor plans are reasonably easy for the layperson to understand because most people are familiar with the drawings representing geometrical objects. The architect and the client share common concepts about space and geometry. But the software engineer must represent for the client a system involving logic and information processing. Since they do not already have a language of common concepts, the software engineer must teach a new language to the client before they can communicate.

Moreover, it is important that this language be simple so it can be learned quickly.