A2.4 THE FUTURE OF SOFTWARE ENGINEERING

Software engineering has come a long way since 1968, when it was first used at a NATO conference. Software engineering got a boost when it responded to the need to build custom-based systems for defence, government and industrial applications. We now develop wider systems such as embedded, Web enabled, intranet and enterprise. These systems cover applications in manufacturing, services, distribution, process control, communications and so on. As software systems requirement started growing from the late sixties, the software developer community as a whole started thinking of automating the processes by developing tools and techniques. When software started becoming an activity performed by different teams, at different locations and on different platforms, it raised the requirement of co-operation, collaboration, co-ordination and communication, and on completion of the development its integration for testing and implementation. Besides cost, risk and quality and technology became critical components in the development effort. The software community believed that a scientific and systematic effort was necessary for successful completion of software development and they sat down to evolve an SE discipline consisting of the following:

- Principles
- Methods
- Scheduling
- Implementation
- Processes
- Models
- Manufacturing
- Learning
- Methods of evaluation
- Planning
- Testing

When requirements become complex and large and users and stakeholders demand continuous changes in the software system, adopting engineering discipline became very necessary for the benefit of the community at large.

Any engineering faculty has the following features

- Standards: They could be universal and/or specific to a domain in all critical areas of the subject.
- Principles: Based on pure sciences and its application for developing subject -related engineering principles.
- Models: Development of different types of models using 'Abstraction' process for viewing, simplifying and prototyping.
- Methods: Standard methods for common processes like analysis, measurement, computing and communications.
- Artifacts: Use of templates, forms, checklists for formalization of documentation and communication for common understanding.
- Tools: Tools for automation, mechanisation, simplification of process to save time and cost.

- Estimation techniques: Estimation of inputs, resources and efforts for costing, budgeting, planning and control.
- Building standard practices: Uses experiences to develop most efficient and effective methods of applications, which also became best practices at a later date.

Software Engineering

Software development effort started becoming large, complex, technology-driven and influenced by human behaviour and environment changes, which required strict controls on quality and cost and timely deliveries. Slowly, it started evolving as an 'engineering' discipline, moving from disorganized non-standard unsystematic activities to goal-oriented, systematic, methodical and scientific activities, using universally accepted engineering principles, concepts and discipline.

Software engineering uses the following extensively

- Standards: Programming standards, documentation standards, protocols, COM/DCOM, CORBA standards, design standards, test standards, GUI standards and so on.
- Principles: Principles of development, testing, implementation where violation is risky and fatal. For example, it is principally wrong to use the waterfall model for the development of a dynamic, open, complex system. To save cost and effort principle is to use reusable component technology.
- Models: Waterfall model, Boehm model, process models, COMOCO model, design models.
- Methods:
  - Analysis – Design – Build – Test – Deliver.
  - Unit testing, module testing, □–testing, □–testing.
  - Installation testing and user acceptance testing.
- Tools: CASE, report writers, editors, interpreters, modeling tools (rational, visio), compilers, DBM tools, CMM models and so on.
- Estimation: Function point analysis, COMOCO II model
- Building standard practices:
  - Use of templates and checklists.
  - Reviews/ inspections; prototypes
  - Technology applications

As software development activity became complex, requiring specialization, management and engineering principles became imperative. Software engineering is not a pure science, but largely an application science with a very tight and close connection to computer science and technology. It is in continuous evolution, tending to become full-fledged computer engineering.

✓ *Wasserman Principles*

Wasserman, stated seven key factors that have altered software engineering practices. The key factors are:

1. Criticality of time to deliver.

2. Change in software development economics, i.e. shift from higher hardware cost to lower hardware cost, indicating lower power performance *versus* cost ratio. Also, shift from low development and maintenance cost to high development and maintenance cost.
3. Change in computing culture: centrally controlled main/mini platform to distributed client server user-driven, desktop computing.
4. Computing focus shift from local to wide area networks. use of Internet and Web technologies.
5. Paradigm shift in development strategy, shift from SSAD to OOSAD technology.
6. Design and architecture style based on user behaviour using use cases and unified modeling language.
7. Very rare use of waterfall model due to its inherent rigid approach to development and shift to Boehm's spiral model with iterative evolution approach to development.

Wasserman pointed out that any one of the seven technological changes would have a significant impact on the software development effort. Together, they have affected the work style and work culture of software development.

Wasserman identified eight fundamental notions in software development that formed the basis for software development becoming software engineering. The eight notions are:

- Abstraction: A process of taking a limited but critical view to understand the problem to find solution. Abstraction helps to concentrate on the problem and ignores the details or entities, which have no impact on the problem.
- Analysis and design methods: Analysis and design methods serve the purpose for which they are made but also provide a medium of communication. They help us to build reusable components, improving productivity and quality of the software. However, in all these methods there is no common method of representation that can be used to communicate to others. Each member and team has their style of representation. So what we are saying is methods are common (insignificant variations) across the team and organisations, but the manner in which they are documented and conveyed are not common, same and standard.
- User interface prototyping: Prototyping is used often to build a good user interface design. But the use of prototype is not limited to design but can be extended to improve function, feature and facilities. User interface quality is greatly affected by user behaviour and expectations helping to build a better user interface.
- Software architecture: Architecture describes all the units of the system or software and how they are connected with each other. Architecture is

built through the process of decomposition of a system/software top-down.

- **Software process:** It is very difficult to prescribe a development process for the software. This is because software differs from application to application and organisation to organisation, calling for different processes of development. The process differs for simple, single user, desktop application when compared to enterprise-wide, multi-user complex applications. Wasserman prescribes tailoring the process to software application.
- **Reuse:** The notion of reuse is to take the benefit of development already made in many other applications and save the effort and cost of current software development. The notion is based on the principle of commonalities across the application. Reuse of code/application /program from other applications in the concerned application reduces the cost of development, improves efficiency, and also the effectiveness of the software.
- **Measurement:** Measurement helps to assess the status of an entity of interest. The same is true in software development. In software, measurement is used to quantify size, cost, effort, quality of the software. Measurement and metrics help to assess, monitor and compare progress among various applications and projects.
- **Tools:** The notion of tools is to automate the process effectively and attain higher quality. However, tools do not address all the requirements of the entire development cycle. There are many tools and their use requires integration in the development cycle and in the development environment.

All the eight notions together integrate the disparate activities of software development and elevate software engineering into a scientific engineering discipline.

Wasserman suggests five ways of decomposing systems into smaller modules or units

1. **Modular:** Function is a module. Sub-function is a sub-module.
2. **External data driven:** Data structure, basis for decomposition. That is, use of data hierarchy constructed by type and application or usage
3. **Event driven:** Event-responding decomposition Event could be external or internal. For example, ATM Card swapping will trigger an event. Receipt of material is an event generating receipt processing.
4. **User inputs driven:** Wasserman calls it ‘outside-in design’. Decomposition is built on the basis of user inputs to the system. For example, at the billing counter in a shop, the package is scanned. A unit is needed to receive it and process it.

5. Class-object driven:	Viewing the system in class and objects and their relation ship.

The five ways are not mutually exclusive.