ISSN:0975-9646

Future Challenges and Opportunities of Systems and Software Engineering Processes

Rohit Sharma [1] ,Samridhi Sharma [2] ,Sachleen Singh [1]
[1]*CT Institute of Management & Information Technology*
Jalandhar (Punjab), India
[2]*Punjab Technical University*
Jalandhar (Punjab), India

ABSTRACT-Software Engineering is a profession dedicated to designing, implementing, and modifying software so that it is of high quality, affordable, maintainable, and fast to build. The impact and importance of software has come a long way. Software engineering is the technological and managerial discipline concerned with the systematic production and maintenance of software products that are developed and modified on time with in cost estimates. And yet, a new generation. The paper starts by examining the past, current, and future states of software engineering. The paper then examines the critical technical issues in software engineering including complexity , structure, and evolution of software systems; economics of software engineering, and measurement of software engineering products and processes, as well as the critical people and organizational issues including learning, motivation and performance improvement. This result can serve as a repository of valuable information for the people who are aspiring to do research in software engineering.

KEYWORDS-Software Engineering, Complexity, Economics, Structure, and Evolution of Software systems.

INTRODUCTION

Though a substantial body of knowledge exists in Software Engineering, a large number of issues are still open or in need of further research. One of the most important issues confronting software engineering research is the identification of fundamental issues, challenges and opportunities for research in the discipline. These must be identified and better understood if we are to significantly improve our practice of software engineering. At the dawn of the new millennium, software managers face a particularly difficult set of challenges. More than ever before, organizations depend upon computer software for their competitive survival. Large, complex, and inter-networked software systems play a critical role in many aspects of organizations' value chains. Users need software that can meet stringent requirements, can be produced quickly and productively, and can be easily maintained to keep pace with an ever-increasing demand for functionality, quality, and cost-effectiveness. The rise of the World Wide Web and electronic commerce have intensified the challenges of software development by dramatically shortening product development cycles and elevating time to market as a critical dimension of software development performance. Software systems developed with high quality, within budget and without delays. Despite pressures for increased productivity, quality and timeliness, and the introduction of major software process innovations, many software projects continue to experience significant schedule delays, cost overruns, and quality problems.

Six Broad Categories Identified That Affect Software Engineering

The broad categories that affecting software engineering were identified to examine the critical technical as well as the critical people and organizational issues.

1. **The Complexity, Structure, and Evolution of Software Systems.**
2. **The Economics of Software Engineering.**
3. **The Measurement of Software Engineering Products and Processes.**
4. **Learning and Improvement in Software Engineering: Individual and Organizational Perspectives**
5. **Software Engineering: Past, Present and Future.**
6. **People-Related Issues in Software Engineering.**

1. THE COMPLEXITY, STRUCTURE, AND EVOLUTION OF SOFTWARE SYSTEMS

1.1 Definition of Software Complexity

Software complexity is defined in IEEE Standard 729-1983 as: "The degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, the types of data structures, and other system characteristics. "Software complexity is a link between development practices and software maintenance performance. High software costs and huge time spent in testing and maintenance of the software makes the software complex and makes it difficult to maintain and modify the software. Software complexity is aimed to objectively associate a number with a program, based on the degree of presence or absence of certain characteristics of software. It is assumed that software complexity is related with such features of software like number of errors left in the software, effort to design, test or maintain a software product, development time, maintenance cost, etc.

1.2 Importance of Software Complexity

Knowing the complexity of a specific software product or its module enables one to:

i. Predict the cost of testing, maintenance, etc., number of errors left in the software, size of the final program.

ii. Assess the development time, effort, cost, etc;

iii. Identify critical modules or parts of the software;

iv. Compare programs, modules, programmers, etc. according to software complexity.

1.3 Management of Software Complexity

Modularizing the system is the most effective way of managing the complexity of the software. Besides, Flow graphs and structure graphs can be used to manage software complexity. Both in the design phase and the implementation phase, modularization has to be stressed.

1.4 Engineering of Software Systems

It is now well known that software system should be engineered for software development is not a self-development process. Software development depends upon various features such as Volatility, Software Structure, Complexity and Enhancement. Engineering the systems will provide us with an estimate of the costs and development time and helps in planning.

1.5 Issues, Challenges and Opportunities

Software complexity is one branch of **software** metrics that is focused on direct measurement of software attributes, as opposed to indirect software measures such as project milestone status and reported system failures.

2. THE ECONOMICS OF SOFTWARE ENGINEERING

2.1 Software Production-Expensive

It is often expensive to produce software because of the following factors:

i. Highly skilled labour is required
ii. Highly developed technology is needed
iii. High maintenance cost
iv. High debugging cost

2.2 Ways to Reduce Software Cost

Software cost can be reduced by development and use of models for cost estimation. Also it can be reduced by estimating the cost and benefits of investments in reducing controllable complexity and in promoting initial software quality. Software must be reused when possible. Software can be customized so that they can serve the needs of all people within the group. In doing so, support and maintenance costs can be reduced, and users can be benefited from having ready access to others who are using the same software.

2.3 Issues, Challenges and Opportunities

Software engineering focuses on the production of software, which is by its very nature a relatively intangible good. Objectifying and measuring its many dimensions are often challenging to researchers. Even in situations where we have relatively good software related artifacts to examine, there may be little evidence to examine concerning the process by which they were constructed. Researchers should be clear about the fundamental ideas within any new technology-technology innovators need to be explicit about why the new technology is believed to work and evaluation research should be pegged to these fundamental ideas, rather than the surface nomenclature.

3. THE MEASUREMENT OF SOFTWARE ENGINEERING PRODUCTS AND PROCESSES

3.1 Importance of Measurement

It is **important to measure in software engineering because if we don't measure, judgment can** be based only on subjective evaluation. With measurements, trends (either good or bad) can be spotted, better and estimates can be made and true improvement can be accomplished over time.

3.2 Aspects/Dimensions to be measured

Software engineering encompasses two major functions of planning and control, both of which require the capability to accurately and reliably measure the software being developed. The various aspects or dimensions of s/w engineering , which required to be measured, include estimation of appropriate budgets and schedules.

3.3 Definitions and Evaluation of Software Engineering Measures

A software engineer collects measures and develops metrics so that indicators will be obtained. An indicator is a metric or combination of metrics that provide insight into the software process, a software project or the product it.

3.4 How to Know a Good Software Measure

We know that a software measure is "good" if:

i. There is correct estimation of budgets and schedules
ii. Good control over progress of software development.
iii. Accurate measures of the complexity-adjusted size of the deliverables of a software project early in the life cycle. This will permit the estimation of the relationships between the deliverables and the cost and time required to produce the software.

3.5 Issues, Challenges and Opportunities

There are many challenges and opportunities in research on measurement in Software engineering as the formal framework for these measures have yet to be defined clearly. With a formal framework comes the challenge for framing the properties for these metrics.

4. LEARNING AND IMPROVEMENT IN SOFTWARE ENGINEERING:

INDIVIDUAL AND ORGANIZATIONAL PERSPECTIVES

4.1 Difficulty for Individuals in Learning

It is often difficult for individuals to learn in software engineering. Lack of understanding of one's cognitive processes during development of software can be major source of difficulty for an individual to learn in software engineering.

4.2 Improving Individual Ability to Learn

Software engineers can increase their ability to learn, improve and innovate through:

i. Awareness: Software engineers can be more aware of the surroundings and the knowledge of related complex technologies should be updated from time to time.
ii. Interest: Software engineers should have keen interest in the complex technologies and Software Process Innovations that are going to be in the organization.
iii. Evaluations/ Trials: Organizations should arrange various workshops and seminars to make user comfortable with the Software.

4.3 Difficulty for Organization in Learning

When complex organizational technologies are first introduced, the distance of learning is likely to be considerable for most organizations. In the case of software process innovations there are various factors affecting the organizational to learn in the software engineering. Successful assimilation requires learning on a number of fronts such as:
i. Including grasping the abstract principles on which the technology is based; understanding the nature of the benefits attributable to the technology.
ii. Grasping specific technical features of different commercially available instances of the technology;
iii. Discerning the kinds of problems to which the technology is best applied;
iv. Acquiring individual skills and knowledge needed produce a sound technical product on particular development projects;
v. Designing appropriate organizational changes in terms of the team structure, hiring, training, and incentives. It is hard to overstate how difficult and expensive this learning process can be.

4.4 Issues, Challenges and Opportunities

There are many issues, challenges, and opportunities in research on individual and organizational learning in software engineering. Organization and individual should learn collectively so as to really implement the innovations and complex technologies in the organization. User should have knowledge of related technologies and company should slowly and slowly bridge the gap ok "knowledge barrier".

5. SOFTWARE ENGINEERING: PAST, PRESENT AND FUTURE

5.1 Problems in the Past

In the past **software** development faced the problems of planning, organizing, staffing, coordinating budgeting and directing the software development activities. The systems were batch processing and so the programmer has to wait for a long time for the program to compile and execute. Also, most of the projects used to run over budget and behind schedule. Most of the times, the problem requirements were not properly understood. There was a lack of proper design and analysis tools available for project evaluation at every stage. The major problems were in the spheres of error detection, defect management, data abstraction, etc.

5.2 Critical Problems Today

An **engineering** approach to the development of computer software is now a conventional wisdom Although the debate continues on the "right paradigm", the degree of automation, and the most effective methods, the underlying principles of software engineering are now accepted throughout industry.

5.3 Critical Issues in the Future

In the future, developing large-scale systems will still be a problem. Systems development will become a process within which needs are formulated and then potential solutions are then selected from a large solution space. The legacy problem is likely to get worse when software consists of diverse components obtained from many different sources. Continual product churn and planned obsolescence may lead to a lack of confidence in the software industry- and a resulting lack of investment by potential users. Software should meet necessary and sufficient requirements, should be personalized, self adapting, fine grained, should operate transparently. End users should develop it, will be component based.

5.4 Difficulty in Developing Software

The demand for functionality, quality, flexibility and cost effectiveness keep on increasing with time and hence these factors have made it difficult to develop
software . Also, in spite of the fact that the software development life cycle has been identified; there is also a need of personnel and team software planning.

5.5 Issues, Challenges and Opportunities

In the future there would be more number of people working on software development, as the dependence of society on software would maintain upswing. Experience indicates that, as the number of people on the software team increases, the overall productivity of the group may suffer. One way around this problem is to create a number of software engineering teams, thereby compartmentalizing people into individual working groups.

6. PEOPLE-RELATED ISSUES IN SOFTWARE ENGINEERING

6.1 Ways to Improve Individual Software Engineers

The performance of individual software engineers can be improved by giving proper guidelines, teaching soft skills, bettering them in personal software process and team software process which are used for disciplined software development processes at both individual and team levels.

6.2 Difficulty in Creating Software Using Project Teams

Often it is difficult to create software using project teams. Creating a winning software team requires more than balancing commitments and resources. Most teams that sustain a winning tradition over many years and many changes in players do so because they excel at attracting, developing, organizing, motivating, and retaining talent.

6.3 Solution to these Problems

For all these problems there are two categories of solution. One is to deploy the "hard" technologies of communication, from e-mail to videoconferencing. The other solution is deploying "soft" technologies, from a "meeting-the-team" start-up workshop to kick-off bonding and circulating a directory of team members and their skills to immersion courses in language training or translating documentation. And finally, "the project leader should know when people's birthdays are and celebrate them, even if only virtually."

6.4 Issues, Challenges and Opportunities

Some of the common people related mistakes that could be studied are undermined motivation, weak personnel, uncontrolled problem employees, heroics, adding people to a late project, noisy and crowded offices, friction between developers and customers, unrealistic expectations, lack of effective project sponsorship, lack of stakeholder buy-in, lack of user input, politics placed over substance, and many more.

CONCLUSION

All software construction involves essential tasks, the fashioning of complex conceptual structures that compose the abstract software entity, and accidental tasks, the representation of these abstract entities in programming languages and the mapping of these onto machine languages with in space and speed constraints. High-level languages, Time-sharing Unified Programming Environments Object-oriented programming, Artificial intelligence, Expert systems, Workstations, Environments and tools are some of the breakthroughs that solved the accidental difficulties. Some of the challenges for software engineering in the future will be Legacy (proper maintenance of the software in cost effective manner), Heterogeneity (to build dependable software which is flexible to cope with heterogeneity of distributed systems on network) and Delivery (to deliver the large and complex with in time with out compromising on quality) The radical view of the future of software seeks to bridge the gap between technology and society. Also, software will be component based and thus components will be customizable and flexible. In future, we will be looking forward to prioritize these challenges by calculating the impact of each challenge on development of software applications.

REFERENCES:-

1. Brereton, P. et al. "The future of software," Communications of the ACM, December 1999,
2. Leveson, N. "Software engineering": stretching the limits of complexity.
3. F. P. The mythical man month: essays on software engineering.
4. L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos. "Non-Functional Requirements in Software Engineering" Springer Berlin / Heidelberg,pp.363-37(2009)