Comparative Software Engineering: Review and Perspectives – Guest Editors' Introduction

Yingxu Wang* and Dilip Patel**

* Centre for Software Engineering, IVF
Argongatan 30, S-431 53, Molndal, Gothenburg, Sweden
Tel: +46 31 706 6174, Fax: +46 31 27 6130
Yingxu.Wang@acm.org

** School of Computing, IS, and Mathematics
South Bank University
103 Borough Road, London SE1 0AA,UK
dilip@sbu.ac.uk

Abstract: Engineering is a set of disciplines seeking solutions for complicated problems and systems that could not be done by individuals. The aim of engineering is to repetitively produce complicated artefacts in an efficient way. This paper describes a set of generic engineering principles and an engineering maturity model. With the engineering principles and model, the nature and status of software engineering are analysed. Interesting findings on what software engineering can learn from generic engineering principles are presented. This paper intends to show the nature, status and problems of software engineering, as well as its future trends, based on the comparative studies between the generic engineering principles and software engineering practices.

Keywords: Engineering, software engineering, engineering principles, engineering maturity model, nature of software engineering, cross fertilisation

1. "To Be or Not To Be?"

To many professionals engineering means systematic planing, teamwork, rigorous process, repeatability, efficiency. Software professionals have been arguing the term "software engineering" and its implication for three decades since Frits Bauer invented it in 1968 [1-2]. Yet, still some fundamental questions remain, such as:

a. Is software development an engineering discipline?

b. Are software developers engineers or craftsmen?

There were completely different assertions and opinions on the above issues of "to be or not to be" that is still confusing the academics, practitioners, and students [3 - 7] in software engineering and in the software industry.

In investigating these fundamental problems, the authors find the myth was caused by a confusion of time in perceiving software development as, or as not, an engineering discipline. The authors' answer to the question whether software development is an engineering discipline is simply 'no.' While more precisely: it is 'not' at present and in the past, and it is going to be and should be 'yes' in the future. Currently, software development is evolving from the laboratory-oriented and all-round-programmer-based practice to an industry-oriented and process-based platform, and software developers are experiencing changes of roles from craftsmen to regulated professionals – the software engineers. The practices of the former are based on personal talents, tastes and art, while those of the latter are based on disciplined processes and repeatable professional activities.

2. What is the Nature of Software Engineering?

Conventional industries are producing artefacts from raw materials via engineering approaches; Software industry is producing software solutions for problems via software engineering. In 1975 Hoare identified four characteristics of software engineering [8] – professionalism, vigilance, sound theoretical knowledge, and tools. These characteristics were quite generic for perceiving software engineering at that time. In a further development in 1996, Wasserman [9] described eight technical characteristics of software engineering as follows:

- Abstraction
- Methods and notations
- Prototyping
- Modularity and architecture
- Lifecycle and process
- Reuse
- Metrics
- Tools and integrated environments

Wasserman's vision mainly covered the technical characteristics of software engineering. Therefore Hoare's view has been seen as a balance to show both the sights of the forest and trees in describing the young discipline of software engineering.

To investigate the nature of software engineering in a systematic way, the authors find there are five categories of characteristics of generic engineering principles, which software engineering can borrow. They are the categories of engineering aim, organisational, technical, managerial, and social characteristics [6]. According to the classification, a key to software engineering is the category of engineering organisation, which characterises the following features:

- Apply systematic processes
- Support co-operative work
- Adopt division of work
- Establish standardisation
- Adopt tools and machinery
- Plan actual schedule
- Optimise resources allocation
- Derive predictable outputs
- Seek controllable quality

These characteristics determine the level of maturity in engineering organisation. All of them are applicable towards maturing the software engineering discipline. To further explore the nature of software engineering, the authors found it is useful to contrast the three-generation definitions of software engineering.

The first definition of software engineering was provided by Bauer [1] more than three decades ago. In his paper Bauer defined software engineering as: "*The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*"

One of the well-accepted second generation definitions of software engineering, by McDermid [3], is as follows: "*Software engineering is the science and art of specifying, designing, implementing and evolving - with economy, timeliness and elegance – programs, documentation and operating procedures whereby computers can be made useful to man.*"

Along with the evolution of software engineering research and practices, the authors find it is useful to shift the concept of software engineering from conventional laboratory orientation to an industrial orientation. This results in a new perception on software engineering [10, 6] as follows: "*Software engineering is a discipline that adopts engineering approaches, such as established methodologies,* *processes, tools, standards, organisation methods, management methods, quality assurance systems, and the like, in the development of large-scale software seeking to result in high productivity, low cost, controllable quality, and measurable development schedule*."

Contrasting the differences between the three definitions of software engineering leads to the distinctions noted in Table 1. Table 1 shows how the concepts surrounding software engineering can be enhanced, particularly in terms of its means, aims and attributes; resulting in a deeper understanding of software engineering.

Table 1. Analysis of Representative Definitions of Software Engineering

No.	Nature	Means	Aims	Attributes of aims
1	A method	Engineering principles	Software	- economy - reliability - efficiency
2	A science and art	Life cycle methods: - specification - design - implementation - evolving	Programme and Document	- economy - timeliness - elegance
3	An engineering discipline	Engineering approaches: - standards - methodologies - tools - processes - organisational methods - management methods - quality assurance systems	Large-scale software	- productivity - quality - cost - time

Some points regarding the perceived nature, means, aims and their attributes in the answers can be made. The first definition proposed software engineering as a method or approach to software development; the second definition focused on scientific methods and art for programming; and the third definition portrays software engineering as an engineering discipline for developing large-scale software in the software industry.

The above discussion shows that there is a need to shift from the conventional laboratory-oriented and all-round-programmer-based software engineering perception to a modern industry-oriented and process-based software engineering platform. This is going to be a significant trend in the organisation and implementation of software engineering in the software industry.

3. What are the Principles of the Generic Engineering Approach?

Engineering approaches and engineering disciplines emerged during the 19th century industrial revolutions. Before the industrial revolution, people produced goods as craftsmen at small or limited scales and they learnt things by doing.

For improving productivity as well as quality, and for lowering the requirements for skills in mass production, a generic industrial engineering approach [11, 12] had been formed as outlined in the following phases:

a. To identify repeatable work processes

b. To identify standard and reusable components of products

c. To adopt division of work (people specialised in a defined role in processes)

d. To equip specialised tools for the roles and processes

e. To recognise management as a profession for organisation of the processes and for
co-ordination of the roles

These key steps of a generic engineering approach form the basis of almost all the existing engineering disciplines. Historically, every engineering discipline in the modern industries has been developed and matured in the same approach: first a kind of art, then a discipline of engineering.

For instance, in the early 19th century and even earlier, watches were produced manually, and so there were no identical watches. At this stage, the watchmakers were characterised as craftsmen rather than engineers. This resulted in low productivity and high price, and one would perhaps need to find the original watchmaker in order to have a watch fixed. In the middle and late 19th century, the industrial revolutions addressed some of the problems and introduced the approach to engineering. Taking the watch manufacturing industry, as an example again, it was a significant achievement when the industry could massy produce watches by machines, and all watches were identical so that parts were interchangeable among watches of the same brand. At this stage, the traditional watch-smiths had become engineers, who were responsible and skilled for one or limited mass production process of watch manufacturing.

This generic industry approach to engineering is also applicable to software engineering, although it has often been ignored in research and practice. Towards establishing an engineering discipline, software engineering is going to repeat the history of industrialisation as we reviewed in this section. The findings are also useful to support the answers in Section 1 for the fundamental questions about software engineering and software engineers.

4. How the Other Engineering Discipline Matured?

By comparing the differences between an engineer and a craftsman in the time dimension as discussed above, we may elicit a generic engineering maturity mode from the history of industrialisation. With the mode of engineering maturity, we are able to examine the status and maturity level of software engineering as an engineering discipline, and to predict its future development.

Looking at the time dimension, engineering is a discipline matured from arts of craftsmen in terms of scale and rigour. While in the professional dimension, engineering is a discipline parallel to sciences, in which engineers are working on technology development and mass production by applying scientific principles and inventions. When asking how the industrial revolutions had changed the traditional individual or family watchmakers, history tells us that the manual watchmakers had disappeared except a few that existed as a special profession. We may also find similar evolution traces in other traditional manufacturing engineering disciplines. This indicates a universal engineering maturity model (EMM) in the industries as shown in Table 2.

In the EMM model for industry disciplines, there are four levels of engineering maturity such as the *emerging, art, engineering,* and *post-engineering* ages. The key characteristics of each level in EMM have been identified as shown in Table 1. Applying the EMM model, we may find some examples of existing engineering disciplines that are already at level 4 – the post-engineering age, such as civil engineering, mechanical engineering, and electrical engineering. Electronic engineering would be at level 3 – the engineering age, since it is still under rapid development within the context of a wide range technical innovation. Biological engineering is an example of those at level 1 – the emerging age of an engineering discipline.

Based on the EMM model, we can predict that software engineering as a young engineering discipline, is going to be matured in the same way: from art to engineering. Checking with the engineering maturity characteristics, software engineering is found to be a good example of a level-2 discipline in the art age, while it is under a transition toward level 3 – the engineering age.

Table 2. The Engineering Maturity Model (EMM)

Maturity level	Description	Characteristics
1	The emerging age	- Being a branch of an existing discipline - Demands in sciences and/or industry have been identified - Common theories and foundations have been formed - A group of professionals has been emerged
2	The art age	- Varying professional practices - Individual stamps and influences both design and implementation - All processes are dependent on personal talent, art and hobby - Work is experience-based and doing by learning - Individual tends to be wizard for everything in all processes - Chasing new methods and/or technologies before their validation has been proven
3	The engineering age	- Adoption of work division - Established processes - Reinforced standards - Stable professional practices - Defined best practices - Well developed theories and foundations - Proven methods and technologies
4	The post-engineering age	- Well defined processes - Well defined standards - Precisely defined professional roles within a discipline - Refined theories and foundations - Refined methods and technologies - Giving birth of new disciplines

5. What is the Status of Software Engineering as an Engineering Discipline?

Analysing the status of software engineering against the EMM model as a reference system, it can be found that software engineering is actually a young discipline, which is located in the art age and is in a transition to the engineering age. Though, there is still some way to go for software engineering to be a matured engineering discipline.

A detailed analysis of characteristics of software engineering at different maturity levels is provided in Table 3. From Table 3 we can see that current software development practices and software engineering education are still located in the age of art and are progressing towards the age of engineering, because the centre of education and practices is mainly craftsman-and-laboratory-environment-oriented. For software engineering to evolve into a mature engineering age there is still much to do as described in the last column of Table 3. Although, historically, it is encouraging to see that software engineering, as a new engineering discipline, has matured to between levels 2 and 3 in just three decades; the other existing engineering disciplines, e.g. civil engineering and etc., have taken hundreds of years to reach their current levels of maturity.

Table 3. Characteristics of Software Engineering in Different Ages

No.	Characteristics of SE in the art age	Characteristics of SE in the engineering age
1	Individual perception on software development activities.	Team perception on software development activities.
2	A programmer, as software developer, is a master of all skills needed for programming.	A software engineer skills for a single or limited development process(es).
3	Final products reflect personal talent, art, hobby and experience.	Final products are based on sound theory, proven methodologies and common practices.
4	A software developer is a person who has multi-roles as of requirement analyst, designer, programmer, tester, and even user.	A software engineer is a person who has specific role in one of the processes as listed on the left.
5	Software product is a personal solution to an application.	Software product is a standard and regulated solution to an application.
6	Programming is personal interaction between the programmer and a computer.	Programming is a group interaction between all roles and processes, including the user(s).
7	Program is written for machines rather than for human being.	Program is written for colleagues involved in all processes rather than only for machines.
8	Programmers are not trained in formal ways, but believe learning by doing.	Software engineers are trained in formal ways and following common rules.
9	Knowledge transfer is hard in programming, and design and implementation of software is regarded as personal experience.	Knowledge transfer is defined and carried out by hierarchical processes at organisation, project, and individual levels.
10	Problems to be solved are limited in small scale.	Problems to be solved are large-scale software development for complicated systems.
11	Program maintenance is relied on the original designer.	Software maintenance can be carried out independently from the original developers.
12	An individual virtually runs a program using one's mental power for software validation.	Software validation is carried out by rigorous architecture design, testing and logical deduction.
13	A programmer is self-sufficient and self-managed for all processes.	Software engineers are mutually related with a defined as well as the pre- and post-processes.
14	Local available of materials, tools, and solutions.	Global available of materials, components, tools, and solutions.

6. What Can We Learn from the Generic Engineering Principles?

The most important principle of generic engineering organisation is 'division of work', or limitation of the roles of a software engineer in the whole software development processes. Considering that in electronic engineering, an electronic engineer is not supposed to be specialised in all application areas of electric engineering: from low to high frequency circuits, from analogue to digital circuits, from real-time systems to home appliances. Similarly, a car engineer is not supposed to be experienced in all areas of car manufacturing and maintenance, such as mechanical structures, engines, transmissions, electronic systems, micro-controllers, petrel, lighting, safety facilities, and bodies of vehicles.

Therefore, 'division of work' is the key for organising an engineering discipline, which is so obvious and so often to be ignored in current software engineering practices. For large-scale software development, what we need is highly skilled software engineers who are competitive for one or limited roles, rather than a person with all-round skills in the software engineering processes. This is what we learnt from the universal principles of industry engineering.

The second fundamental issue we learnt is that, obviously, there are significant gaps in many important aspects of software engineering, such as:

- team perception
- programming for colleagues in related processes rather than for machines
- following common roles rather than personal hobby
- roles and best practices are explicitly described and regulated by processes rather than remaining as personal experience and private knowledge

- test and maintenance are carried out independently from original developers
- software engineers are prepared to fit in specific process rather than tend to be a master of all-round activities in development
- maximum reuse of available components and tools rather than tends to be self-sufficient.

The EMM model can be taken as a reference for analysing and organising software engineering for a maturing engineering discipline. By clarifying the current status of software engineering as a discipline in the age of art, responsibilities of software engineering researchers and practitioners are to push forward software engineering to a matured engineering discipline by applications of the generic engineering principles we gained from other engineering disciplines.

Note from Ruzanna Chitchyan: section 7 is not relevant, it is for specific articles that we are not reading.

~~7. What are Presented in this Volume on Comparative Software Engineering?~~

~~In the above sections, we presented the EMM model and contrasted characteristics of software engineering in the engineering and art ages. With these results as a framework, we present two sets of articles on "Pinnacles of Software Engineering Technologies" and "Approaches to Software Engineering" in this special volume of Annals of Software Engineering. As shown in Table 4, the papers are selected from contributions of 14 outstanding researchers and practitioners in software engineering among 34 submissions.~~

~~**Table 4. The Structure of ASE Vol.10 on Comparative Software Engineering**~~

~~No.~~	~~Pinnacles of Software Engineering Technologies~~	~~Approaches to Software Engineering~~
~~1~~	~~*D. Bjorner*~~ ~~Pinnacles of Software Engineering: 25 Years of Formal Methods~~	~~*W. Humphrey*~~ ~~Software – A Performing Science?~~
~~2~~	~~*F. Barbier and B. Henderson-Sellers*~~ ~~Object Modelling Languages: An Evaluation and Some Key Expectations for the Future~~	~~*A. Bryant*~~ ~~Metaphor, Myth and Mimicry: The Bases of Software Engineering~~
~~3~~	~~*A. Mili et al.*~~ ~~A Comparative Analysis of Hardware and Software Fault Tolerance: Impact on Software Reliability Engineering~~	~~*B. Beizer*~~ ~~Software is Different~~
~~4~~	~~*C. Rolland*~~ ~~From Conceptual Modelling to Requirement Engineering: an Engineering Approach~~	~~*C. Lewerentz and H. Rust*~~ ~~Are Software Engineers True Engineers?~~
~~5~~	~~*B. Boehm et al.*~~ ~~Software Development Cost Estimation Approaches – A Survey~~	~~*D.J. Ram et al.*~~ ~~An Approach for Pattern Oriented Software Development Based on a Design Handbook~~
~~6~~	~~*J. Tsai and K. Xu*~~ ~~A Comparative Study of Formal Analysis Techniques for Software Architecture Specifications~~	~~*G.A. King*~~ ~~Quality Technique Transfer: Manufacturing and Software~~
~~7~~	~~*D.C. Rine and N. Nada*~~ ~~Three Empirical Studies of a Software Reuse Reference Model~~	~~*B. Banerjee*~~ ~~Mapping Software: Are We Nearing Standardisation?~~

~~We hope that readers of Annals of Software Engineering will benefit from the papers presented in this Special Volume on Comparative Software Engineering and be able to share the cutting-edge research results with the authors. We would like to thank the reviewers for their enormous effort in helping to improve the quality of this special volume that covered a wide range of fundamentally significant issues in software engineering.~~

References

[1] Naur, P. and Randell, B. (eds.) (1969), *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*, NATO.

[2] Bauer, F. L. (1976), Software Engineering, in Ralston, A. and Meek, C. L. (eds.)*, Encyclopedia of Computer Science,* Petrocelli/Charter.

[3] McDermid, J. A., ed. (1991), *Software Engineer's Reference Book*, Butterworth-Heinemann Ltd., Oxford.

[5] Sommerville, I. (1996), *Software Engineering* (5th edition), Addison-Wesley, 1996.

[6] Wang, Y. and Graham, K. (2000), *Software Engineering Processes: Principles and Applications*, CRC Press, USA, ISBN: 0-8493-2366-5, pp. 1-746.

[7] Pressman, R. S. [1992], *Software Engineering: A Practitioner's Approach* (3rd ed.), McGraw-Hill International Editions, 1992.

[8] Hoare, C.A.R.(1975), Software Engineering, *Computer Bulletin*, Dec., pp.6-7.

[9] Wasserman, A. (1996), Toward a Discipline of Software Engineering*, IEEE Software,* Nov., pp.23-31.

[10] Wang Y., Bryant A. and Wickberg, H. [1998], A Perspective on Education of the Foundations of Software Engineering, Proceedings of the 1st International Software Engineering Education Symposium (SEES'98), Scientific Publishers OWN, Poznan, pp.194-204.

[11] Marshall, A. (1938), *Principles of Economics*, The Macmillan Co., London.

[12] Kuhn, T. (1970), *The Structure of Scientific Revolutions,* The Univ. of Chicago, Chicago.