Welcome Message from the General Chair

On behalf of the organizers of this international conference, we extend a warm welcome to all of our participants. This conference has evolved from previous, very successful conferences and we all fondly hope that we all will learn, share, and creatively interact with our distinguished participants for everybody’s mutual benefit and education. We congratulate our steering committee chair, Professor Tse, for providing us with dynamic leadership and our co-chairpersons of the Program Committee, Professors Mathur and Wong, for organizing a great technical program. Our sincere thanks go to the Tutorial Committee for selecting very appropriate and timely tutorials. Our sincere gratitude also goes to the Publication Chair, Dr. Lau, and members of the Arrangements Committees for their wonderful efforts.

We gratefully record our warm appreciation to our distinguished keynote speakers, Professor Edward Lee of University of California, Berkeley, and Dr. Solom Heddaya of Microsoft Corp., for participating in this international conference. We also record our thankfulness to the distinguished Plenary Session Chair, Professor Kothari, and the distinguished Plenary Session speakers, Dr. Nelson Ludlow, Dr. Ray Paul, Professor Philip Sheu, and Professor Stephen Yau.

Software engineering is an application-driven discipline and since it caters to the needs of several disciplines it assumes a transdisciplinary structure, and software quality depends on the interactions of the visions that these disciplines carry.

The quality of software is an aspect that is very hard to define or to characterize. It is like beauty, which resides in the eyes of its beholder, and in our case, quality of software would reside in the “eyes” of the ultimate user and the various stakeholders. It always includes some fundamental ingredients such as reliability, dependability, ease of use and understanding, modification, and maintenance. All of these characteristics and many more are required to ensure a trouble free and friendly operation of a software system. It is at once a static phenomenon as it emanates from the designs of the developers, on which the users have no influence and it is also dynamic in the sense the users experience pleasant and pleasurable moments with the long time use of the system.

Software by its very nature controls and executes a mission ordained by its application. In that sense it embodies the principles of the application discipline as well the principles of the information technology, and in particular software engineering.

We never think of software quality unless we are not satisfied with the software system—in other words, its lack of quality. It reminds us of the great Ben Franklin’s wise saying, “When the well is dry, we learn the worth of water.” Equivalently, think of the oft-quoted comment of Warren Buffett, “We only think of oxygen when we do not have it.”

At this conference, we will be always thinking of software quality: how to create it, how to enhance it, how to nurture it and how to radiate it across process, system, service, and application boundaries and interfaces.
‘Software Quality Consciousness’ (SQC) started some forty years ago, primarily with the U.S. Department of Defense with the advent of the computer controlled ballistic missile systems. This marked the first phase of the software quality evolution. It then spread into the computer communication systems of AT&T, the nuclear reactor control and safety systems. Later it spread its wings over power distribution grids, air traffic control, etc. The first phase of SQC embraced the real-time, nationally critical infrastructural applications and their resources. We are fully aware how malfunctions and outages created large scale disasters such as black-outs and brown-outs in electric power and telephone system, and breakdowns in national and international airline traffic. It is good to remember that this phase of application software was developed and operated by application specialists and computer experts. The hallmark events in this phase are the successes of the U.S. Army’s Safeguard Ballistic Missile Defense system and NASA’s successful Apollo 11 moon landing on July 20, 1969, all under the real-time control of software technologies. We can rephrase Neil Armstrong’s famous words as “One small step for Man and one giant step for Technology.” The quality criteria in these applications were reliability, correctness, real-time criticality, and fault tolerance. The applications were associated with the critical national infrastructures, like military defense, communications, power, and transportation.

The second phase of SQC came into place when computer specialists with programming savvy created and developed business and financial applications on the nascent information technology comprising mostly the mainframes and minicomputers. This second phase was a harbinger of elegant programming languages and methodologies, initiated commercially available software tools to support good programming practices and to improve software quality. The quality criteria in this phase was driven by “standardized” programming methodologies that emphasized high dependability (degree of correctness), documentation, and ease of use, modification and maintenance. The application spectrum in this phase included primarily business and financial applications and university research.

The third phase of SQC starts with the advent of the Internet, PCs, e-mails, servers, intranets, and local and remotely accessible databases, with more and more applications becoming automated on computers. During this phase, we see the information technology become not only a national infrastructure but also a commodity, with people freely using PCs, mobile phones, the Internet, emails, search engines, maps, and databases. The users generally did not have technology backgrounds or computer skills. The driving quality criteria in this phase required that the developers use standard components that could be assembled into larger systems and could also work on standardized platforms, consisting of operating systems, languages, and tools. This enabled the creation of new applications with interoperability on devices, and communication media. Because the information system has become a competitive commodity item, its product and service costs became an important factor.

The fourth phase revolves around the Web (Web 2.0), and its services, emphasizing social networking, entertainment, games, plus digital library access and health care applications. Thus information technology commoditization has entered users’ personal and social lives providing remote but beneficial interactions amongst them with learning and entertainment opportunities. It has invaded our living rooms through entertainment, music and games and its imminent take over of the household chores such as security surveillance, remote control of household appliances, etc. The dominant quality criteria here include security and privacy surveillance, and protection of customer information assets.

Each successive phase of SQC indicates the expansion and extension of information technology into more sophisticated and complex applications, but accessible to more and more users with less and less sophisticated computer skills. This is achieved by well packaged and “standardized” software
and easy user-interfaces. The applications have become also more personal and it is demanding convenience, security and dependability on behalf of its users. We also notice that software quality has an additional mission of making sophisticated computer applications easily accessible, understandable and usable by laypeople with marginal computer skills.

We anticipate many new phases of SQC. We anticipate that many killer applications may be on their way in the future, due to the holy convergence of computers, communications, networking via the Internet, media, mobility, and knowledge bases. Convergence is also occurring in services and systems as it is between humans and machines. Because of the complexity of applications created by such convergence, the quality of service may suffer unless properly designed and managed.

Information technology is still far away from satisfying several of our important needs. Consider software-controlled personal appliances. They should be adaptive to user preferences, variations in habits and experiences, and, most importantly, provide very convenient and helpful support to lay users and to the disabled and disadvantaged users. Users can adjust their appliances to their personal needs based on preferences, fads, fashions, conveniences, and disabilities.

As we said in the beginning, it should be a pleasant, pleasurable, and enjoyable experience to use a quality system. These customized quality supporting functions will be web based, serving online with real-time access. These would use very similar methods to those we currently utilize to maintain and update software systems online but under the user's control. These quality enhancing functions can be a part of the system's dynamic quality aspects. The usual static characteristics of quality would be designed into the system and the user may not have much control over them.

Experience indicates errors creep when a system is modified. Intrinsic errors are also introduced when the system is assembled from off-the-shelf components obtained from different vendors. Since components are developed initially for different applications on different platforms, they probably carry conflicting assumptions that their documentation may fail to reveal. Errors due to such occurrences are difficult to isolate until they unfold themselves with failures. Truth function analysis may be one way of mitigating such instances. Our conventional top-down (e.g., waterfall) paradigm in software development consists of a series of transformations and each transformation introduces some “entropy” (unstructuredness or complexity) into succeeding stages at the expense of simplicity, which is often considered to be an important attribute of quality. Studies have shown that validation and verification stages of software development focus more on high-level application aspects rather than the low-level residual execution time environment. Another issue is the unanticipated behavior due to “emergent” properties (Herbert Simon), when the system is assembled from components with known attributes, but that create unexpected conflicting interactions during execution.

We are still groping to find an algorithm that will indicate when one could stop testing. The use of formal methods and correctness proofs is very helpful, but such methods are limited in their span and scope and much remains to be done to make them useful through automated tools.

In the area of multi-vendor developed component technology, we see a large amount of diversity (shall we say, uncertainty) in the programmer skills, methodologies, application conventions, development tools used, and in their underlying assumptions. The diversity may be overwhelming to create an acceptable structure for standardization.

Coming back to the software based communication and power grid disasters; it is well known that the major causes were failures in their infrastructural resources and their connectivity. Under certain conditions, ripple effects can create dominos or tsunamis provoking substantial disasters in the
process. In our recent study, we considered the crash of a computer program as an instance of a disaster and have used conceptually analogous techniques used in power and telephone and natural disasters for the purpose of “disaster hardening,” which can include security and fault-tolerant defensive shields. The purpose of these methods is to prevent or mitigate the damage that can accrue due to a possible disaster by taking proactive precautions. Security and privacy issues overwhelm the soft quality topics and we have several sessions devoted to these topics at this conference.

While these areas are rich for research exploration, one should also look aside to other disciplines and see what they have done to enhance the quality of their technology and work. The methods developed by Taguchi of Japan for developing high-quality electronic products that emphasize optimization of upstream developmental stages and the ideas developed by Bhide of Columbia University focusing on “polishing up” or shrink wrapping of the downstream stages to enhance consumer convenience and enthusiasm may be very fruitful areas for further study in quality enhancement research. Or as Mr. Lee Kun-hee, CEO of Samsung Electronics has recently said, quality should be in “design, software and finishing touch.” This polishing up and giving the finishing touches to the product or system or service could be done by providing the quality conveniences to the end users that they could control, individually and comfortably. Let us not give them lemons because they may not know how to make lemonade.

C. V. Ramamoorthy
University of California at Berkeley, USA