FROM PROGRAMMING TO SOFTWARE ENGINEERING

The above is the title of a series of presentations that I have made in numerous U.S. and International venues. These presentations have been geared to exploring the "lessons learned" from 50+ years of attempting to build software systems. They have also addressed some of the cultural, technological, and conceptual issues related to building acceptable systems in a timely and affordable way.

A major hinderance to progress in past times seems to be that few, if any, have attempted to revisit the set of design tasks from an engineering perspective and/or initiate a public debate. Rather we have attempted to dress up the loosely and/or de-coupled subtasks with "process management tools" and a variety of independently developed tools. While much progress has been made as to fundamental understandings the focus of research has often been on limited aspects of the various subtasks without explicitly addressing the need for the integration of the bits and pieces of an overall solution.

In addition to the above There is strong evidence to support the idea that several of the system design tasks transcend human abilities to accomplish in an acceptable fashion. Specifically the requirements elicitation/validation tasks would seem to need semi-automated formal machine-based support; in addition the task of creating both the requirements and the design documentation seem to be beyond our best manual efforts. These documents are critical for several reasons including the need to "evolve" the system over time when the application changes. It is interesting to note that we still talk of "software maintenance" in spite of the fact that a more cogent concept has always been "evolution"; the fact that it is "evolution" rather than "maintenance" immediately suggests that you need acceptable documentation. Yet most of us would admit that we have been able to meet this need.

As to the scientific progress that is being made there is much to be done in providing "engineering-level" interfaces. While the next 20 years, or so, will lead to training up the next generation of "system-designers" and/or system/software engineers these folks will be system/software engineers and not academic researchers. Hence, the arcane methodologies that provide the basis for the engineering discipline must be made virtually transparent via appropriate engineering interfaces.

I think it is important to continue to stress the need for scientific research (via computer science, mathematics, etc.). However, we need to be clear about the fact that any research advances are the necessary precursor to the development of a complete set of integrated methodologies/technologies to support the engineering tasks. Importantly, in my opinion, the technologies needed to support a principled engineering discipline (which is certainly not computer science or programming per se) have yet to be well, or completely articulated.

Given all of the above, there is the strong suggestion that the research and development community should re-evaluate the overall design task(s), and past efforts. Lessons-learned, and the understanding that the seductive linguistic de-coupling of the overall task does not faciltiate an acceptable and principled composition of the sub-task products. Therein lies a major scientific and engineering problem.

I have been, and am now, involved with the international research community in articulating a "critical Path" research agenda to eventually enable a principled systems/software engineering discipline. In this context the focus is on describing the perceived "technology needs" based on 50+ years of the "programming paradigm" experience. It should be emphasized that the suite of methodologies that will eventually provide the integrated semi-automated system/software engineering environments are largely the subject of on-going research. However, while we may not understand how the problem(s) will be solved, it is clear to a studied effort as to what the technology needs would seem to be. As examples, not only do the requirements elicitation/validation and documentation tasks require technological advances, the semi-automated translation of any "application-domain language (also legacy systams)" to the analysis/design language of given engineering design platforms needs to be addressed as does the iterative issues of "natural language to requirements to specifications".

In the past several years there has been a growing interest in articulating a complete (end-to-end; natural language requirements to fielded systems) paradigmatic concept. In essence there is a growing consensus as to the need for an integrated, interactive, and semi-automated suiteof tools or a platform/environment to support the engineering needs. It may be useful to have a full consensus as to the elements of that future platform; however, that may not be achievable and/or desired. Nevertheless, it is important to motivate a discussions leading to groups of researchers having some strong agreement as to the future configuration of design platforms. Namely,investigators are beginning to realize that the merits and utility of a given theoretical advance lies primarily with its potential for clean integration into an overall context, and to consider this in the "crafting" of the partial solutions.

To prompt useful scientific discussions, facilitate international collaborations, leverage investments, and prompt integration ARO has supported an on-going series of international invitation- only workshops. These workshops have often been co-funded by NSF,DARPA,AFOSR, and ONR. These workshops have been an element in the strategy of addressing all the concerns above. In addition a number of the investigators in the ARO research program have been working to create a complete set of sub-tasks that spans the scope of the engineering system/software design needs; these needs(subtasks) are then being used to evaluate the completeness of the several nascent platforms presently under development. As a second activity we are mapping the various "tools" that are available against the list of needs. The workshops, and the two activities having to do with articulating the elements of a complete solution and an evaluation of tools, has led to significant productive discussions, some international collaborations, the integration of some methodologies, and most importantly (for the on-going research efforts and defense of future investments) some agreement on the elements of a critical-path to the future.

Some Thoughts For Inspiring Discussions and "Out-Of-The-Box" Thinking

Describe a "visionary" (20 year timeframe) end-to-end "solution" for generalized principled software engineering. Note, proposed "vision" should address all software life-cycle lessons-learned in a way that would lead one to developing rational expectations for successful revolutionary implementation. Using your "vision", articulate the critical-path fundamental research areas. All aspects of your "vision" should consider the potential contribution of any "sub-tasks"/methodologies to the idea of "value-added"- schedule, cost, quality, and the sustaining (evolution) of the system.

Analyze the problem of effectively competing for research resources. It is suggested that part of the problem is the large influence of the "status quo". Another part is thought to be the lack of "credibility" (as to the potential ROI on investments) when proposed new efforts are articulated as refinements to largely unacceptable traditional methods and/or are stand-alone partial solutions. A further perception is, that in the absence of an academic/ professional software/system engineering discipline, the din of opinions and practices prevails and undermines the confidence needed to build institutional support for continued R&D. Given the above, and any other issues you perceive, how can we address the need to successfully justify continued research (with the possibility of enhanced funding) while maintaining the sincere belief that the recommended research directions will make a difference.

So-called requirements-engineering needs would seem to beg for revolutionary methodologies for the rapid (and often collaborative/interactive) capture of perceived/desired requirements, rapid iteration/refinement of stated requirements, machine-based "sanity-check" of requirements (for completeness, coherency, etc.), generation of requirements-documentation, and a machine-compatible output available as input to the next stage of the design process. What ideas do you have to exploit past work, facilitate integration of the various approaches, and give direction to future research efforts?

Software system maintenance, which should more rationally be referred to as system evolution, requires that system design- documentation be complete and accurate; however, there is strong evidence that humans cannot adequately/acceptably document the systems they design. Conceptually, this problem may be overcome if we could semi/automatically generate the documentation as an artifact of the design process. Do you have any have any thoughts as to dealing with this idea?

It is expected that some of the present, and newly initiated, research efforts will (in principle) result in potentially remarkable ROI; however the immediate transition to practice will be hampered by the level of expertise required to effectively use these new techniques and integrated suites of tools. This would appear to suggest some ideas for focused research, and the need for a concern for the development of adequate academic curricula for a software/systems engineering discipline. As to the research, human-computer interfaces are needed to support the practice of software/systems engineering ; the complex (and maybe arcane) methodologies needed to support the various tasks should be largely invisible to the average practicioner. Simplifying the high-level implementations of fundamental methodologies with appropriate interfaces is only a partial solution, there is need to eventually transform the workforce to an engineering community schooled in new techniques. Can you suggest how we treat of both the "interfaces" and educational challenges.