SOFTWARE ENGINEERING EDITORIAL

Bernd J. Krämer

Department of Electrical and Information Engineering
FernUniversität, Hagen, Germany

This special issue is dedicated to the memory of Herbert A, Simon, in recognition of his research, his mentoring of a generation of junior researchers, his leadership of the Society for Design and Process Sciences and Software Engineering Society.

It is now 35 years that the "software crisis" was declared and the term „Software Engineering" was coined by a Nato study group. The claim of this group was that the design, implementation, and evolution of software can be organized as an engineering-like activity (Naur, Randell, 1969). Since then the field never lost its attraction to researchers and funding agencies but rather grew to an extremely broad discipline including many aspects from computer science, mathematics, management, economics, and even psychology. It has nearly taken that time to establish software engineering as an accredited engineering discipline, a licensed profession, and an academic degree, which can be earned now at more than 70 universities in Australia, Europe, and North America. The body of knowledge in this field, which has accumulated over the past, is currently documented by the international SWEBOK[1] project group initiated by the IEEE.

In the early days, research and development in this area focused on pragmatic techniques and software engineering environments aimed at a coherent set of tools that supported all technical and managerial aspects of software development processes. At that time "process" designated a vague concept, while the software product was the topic of concern. The importance of the emerging focus on process was first documented in the theme of the 9th International Conference on Software Engineering held in Monterey 1987. This event stimulated a rapidly growing sub-discipline, which generated its own conference series, journals, and monographs.

Software design was a theme of continuous interest and progress in software engineering, which ultimately culminated in a rich set of theories and practices for designing and evolving software architectures. Design issues at this abstraction level address the gross organization of complex software systems as consisting of high level computational building blocks, called components, and their interconnections. This birds-eye view of software applications provided the footing for a new, component-

based approach to software engineering. Building software with reusable components promises to provide many advantages. Whether they can really be achieved like in other engineering disciplines is still to be seen. Research and development are still at an early stage and the long-term consequences of the new engineering approach on software development practice are still unclear. Key questions emerging from this subfield include the design and maintenance of components for reuse, the flexible and impact-free exchange of components in existing systems, and the targeted adaptation and evolution of components.

This special issue is opened with Herbert A. Simon's visionary view on the "Role of Software Engineering in Systems for Design and Process Control," which he communicated in the form of a video speech at the Fifth World Conference on Integrated Design and Process. Starting with a reflection of an earlier message to the SDPS community, Simon strongly argues in favor of designing understandable and evolvable systems. Software evolution is a recent development paradigm that recognizes the economic extent of companies' and organizations' software investments and the long-life of commercial software applications. It views the initial software development activities, which are often confused with the overall process, just as the beginning of a possibly unlimited process of continuous change, where each step in this process must not invalidate the qualities of the actual software system. The core objectives each software engineer should bare in mind includes usability of both the application system and the engineering methods and tools, learning and reflection of software tools, and a component view of software.

You will be surprised to see to what extent the next six contributions address different facets of the claims raised in Simon's message communicated at a time when these papers weren't even written. A common thread is the emerging paradigm of component software engineering but each paper addresses a different facet including evolutionary development of software architecture, design-by-contract enriched by a notion of trust, requirements elicitation, software internationalization, simulation of distributed real-time applications, and visual support of component developers.

The paper by Sünbül, Weber, and Padberg emphasizes the need for continuous software change driven by evolving business processes. Their approach relies on a component-based software architecture. This architectural framework includes both computational components, which provide services to their users, and business process components, which represents parts of business processes and are linked to service components. Hence any change in organizational behavior that is documented in a revision of corresponding business process components may automatically trigger a change in the service components affected. This approach contributes to Simon's claims of understandability and evolution.

Schmidt, Poernomo, and Reussner, in their article "Trust-By-Contract: ..", develop novel architectural mechanisms that allow designers of distributed, component-based architectures to enrich the contractual specifications at component interfaces for the sake of compatible component replacement while maintaining architectural stability. Their notion of protocol rules bares many similarities with the business process components of the previous paper. Besides supporting evolution control, their focus is on enhancing trust in components for third-party users.

Understanding and observing the needs of the contractors and users of a software system is a core research issue in software engineering for about a decade now. A major challenge to this research is the gap between largely informal and visual techniques that are used to capture requirements and communicate initial system designs to stakeholders and rigorous modeling and specification techniques preferred by software developers. The paper "Automatic Synthesis of Behavioral Object Specifications from Scenarios" by Khriss et al. tries to bridge this gap by using a technique for requirements elicitation and specification building that is popular in practice but lacks some rigor necessary for technical development steps. They propose to document scenarios, which capture only partial descriptions of system behavior, with use case and collaboration diagrams of the Unified Modeling Language (UML). They suggest a four-step process for synthesizing behavioral specifications from such scenario descriptions. This contribution addresses

Simon's quest for understandability and usability of development methods and tools. Their algorithms are ready to enhance the functionality and use of commercial CASE tools supporting the UML.

In his criticism of the current state of software development, Simon raised the issue of increased international communication. His concern is picked up and treated in much detail in "A Practical Look at Software Internationalization" authored by Purvis et al. This contribution tackles cultural, linguistic, and technical problems of software to be used in different parts of the world. The authors illustrate their findings and their cure with a practical example of a Java-based tool supporting discussion and decision-making in geographically dispersed teams. They conclude with a decent analysis of the impact of internationalization on the entire software development process.

Alfert's and Fronk's paper on "Experiences in 3-Dimensional Visualization of Java Class Relations" dive even deeper into the software development process and present a new tool that helps software developers to maintain a better survey of the plethora of interconnections between software components, made precise with the example of Java class relationships. They have implemented a novel class browser that supports 3-dimensional representations of class structures for Java applications. Color, geometry, and shape are exploited to visualize different semantic concepts and thus enhance understandability and usability of software design tools.

Kim's paper on "Multi-Level Distributed Real-Time Simulation Based on TMO Modeling" introduces two new perspectives on software: real-time requirements and behavioral simulation. Simulation is an important technique to obtain a proof-of-concept prior to investing a lot of money and man-power in coding activities or to enable cost-effective testing of software systems outside their operational environment. Major advantages of the new simulation scheme proposed in this paper include improved capabilities for real-time software evolution and component-orientation.

I encourage you to read these illuminating articles and start a lively debate with the authors.

References

Naur, P., and B. Randell (eds.): Software Engineering: Report on a Conference Sponsored by the NATO Science Commission, Garmisch-Partenkirchen, Germany, 7–11 October 1968.