1. Introduction to the Minitrack

Software architecture is a burgeoning field of research and practice within software engineering. Or, to be more precise, the architecture of large, software intensive systems has been the subject of increasing interest for the past decade. What accounts for this surge of interest in a field that, until about 1990 was unheard of?

To begin, the field did not spontaneously create itself in 1990. But that time-frame was when the term “software architecture” began to gain widespread acceptance and when the field first attracted substantial attention from both industry and the research community. The field was created out of necessity. Software systems were growing larger: systems of hundreds of thousands or even millions of lines of code were becoming commonplace. Clearly the foundations of the ideas underlying the field that is today called “software architecture” were laid by Parnas, Brooks, Dijkstra and others in the 1960s through the 1980s—but what changed is that by the 1990s such large systems were becoming common. The pervasiveness of such huge systems meant that they needed to be understood and managed by ordinary skilled engineers. These systems were no longer solely the province of a select few virtuosos.

Fundamentally, there are three reasons why software architecture is important to large, complex, software intensive systems: 1) Enhanced communication among stakeholders. Software architecture represents a common high-level abstraction of a system that most if not all of the system’s stakeholders can use as a basis for creating mutual understanding, forming consensus, and communicating with each other. 2) Early design decisions. The software architecture of a system is the earliest artifact that enables the priorities among competing concerns to be analyzed, and it is the artifact that manifests the concerns as system qualities. System tradeoffs are all manifested in the architecture. 3) Transferable abstraction of a system. Software architecture constitutes an intellectually graspable model of how a system is structured and how its parts work together; this model is transferable across systems promoting large-scale reuse.

This minitrack contains 9 papers that collectively address each of these critical issues. Grundy’s paper “Software Architecture Modelling, Analysis and Implementation with SoftArch” describes an approach to creating an analyzing software architectures that emphasizes successive refinement of architectures and provides tool support to analyze them. Vasudevan and Landis’ paper “Mobile Agents for Managing Wide-Area Telecommunications Systems” describes a generic architecture for managing communication systems via mobile agent technology. Khare et al contributed a paper entitled “xADL: Enabling Architecture-Centric Tool Integration with XML” that describes an architecture description language (ADL)-neutral interchange format that allows sharing of architecture descriptions among tools. Herring and Kaplan’s contribution, entitled “The Viable System Architecture” describes their control-based approach to managing what they term “complex systems”, (smart environments, ambient computing, multi-agent systems, and adaptive/intelligent user interfaces). Leitch and Stroulia’s paper “The Space Station Operations Control Software: A Case Study in Software Maintenance” describes how the maintenance of a complex architecture was analyzed and managed. Woodside’s paper, “Software Resource Architecture and Performance Evaluation of Software Architectures” introduces the notion of resource architecture and presents a model of resources and abstractions such as resource-operations that seek to bridge performance evaluation models and software architecture models. Johansson et al have contributed “The Importance of quality requirements in software platform development - A Survey”, in which they present a survey of quality requirements affecting an architecture and how these are prioritized according to often conflicting factors. Issarny and Banatre’s paper “Architecture-Based Exception Handling” describes a formal model and tool support for systematically adding exception handling support to architectures. Finally, Wijnstra’s contribution “Quality Attributes and Aspects of a Medical Product Family” describes how a consideration of quality attributes in the problem space and aspects in the solution can structure the architecting process.