The field of software engineering originally concentrated on “programming in the small” emphasizing implementation techniques. Over the last decades, software and systems engineering expanded to a comprehensive engineering discipline. Nowadays software engineering comprises requirements engineering, modeling and design, implementation and testing, installation and maintenance. Software engineering proved to be a key technology; its mastery determines the functioning of modern infrastructures and consumer products.

Software development has matured from heuristic practice to an engineering discipline. In the meantime, software engineers can benefit from a solid stock of basic research addressing modeling and design techniques for sequential, concurrent, distributed, and timed systems. Software quality and reliability, security and safety, the management of change, reuse and integration pose new requirements to the practice of software engineering. A major challenge consists in integrating the developed theories into a coherent engineering framework accepted by practitioners.

Software engineers face different system views described by a variety of graphical and textual formalisms. Software development must safely bridge the system views on different levels of abstraction ranging from high-level specifications to executable code. The system views center around the data model, the communication model, the state transition model, and the trace model.

The data model describes the data structure in an abstract or concrete way. The communication model manifests how a component interacts with the environment by exchanging messages. The state transition model describes an abstract implementation where an input affects an update of the internal state and a possible output. The trace model characterizes the operational behaviour of the system by a set of possible runs composed of atomic actions.

The lecture classifies different system models and addresses discrete interactive systems with asynchronous communication in greater detail. We present a model for interactive components along with transformations refining the component's behaviour, interface and state. The approach supports the design of elementary components for processing, memory, control and transmission. Moreover, it allows a structural and behavioural description of component-based software architectures and services.

Finally we relate the approach to pragmatic engineering techniques focusing on UML-oriented state transition diagrams and sequence diagrams.