Software has become the key asset for competitive products and services in all industries. Thus, competitiveness in software development, maintenance, and related services has become a concern for organizations. Competitiveness can be increased through (1) internal strategies such as the strategic creation and reuse of software assets and (2) external strategies such as outsourcing software development, maintenance, and/or services from third party service providers and acquiring off-the-shelf components from providers and open source communities. A viable third strategy is to enact both strategies in parallel. This minitrack focuses on the first and third strategy.

Software product line engineering (SPL) is an industrially validated methodology for developing software-intensive systems and services faster, at lower costs, and with better quality and higher end-user satisfaction. It differs from single system development:

1. It needs two development processes to work optimally: domain engineering and application engineering. Domain engineering defines and realizes the commonality and variability of the product line by establishing a common software platform. Application engineering derives applications by exploiting the commonality and binding variability built into the platform.

2. It needs to explicitly define and manage variability. During domain engineering, variability is introduced in all assets such as requirements, architectural models, components, and test cases. It is exploited during application engineering to mass-customize applications to the needs of customers.

Software product line research has mostly focused on the modeling and management of variability in the context of embedded systems (e.g., cellular phones). Most software product line experiences have been obtained from large government and private organizations. This minitrack welcomes contributions from the mainstream product line research. It also acknowledges that the extant body of knowledge in the field is fragmented. More holistic and integrative research approaches are needed to help practitioners leverage the research results in establishing and improving software product lines. Indeed, experienced practitioners have sometimes established innovative product lines and enabling practices and systems with limited awareness of the software product line body of knowledge.

In 2012, this minitrack consists of two papers that address the critical role of integrated information systems in supporting the end-to-end system development life cycle.

The paper of Lu and Käkölä takes as its starting point the fact the development activities of software companies are typically distributed in different places and involve multiple stakeholders in different countries. Testing is an important way to ensure product quality but it is costly and takes ample time and resources. Sourcing testing services from specialized service providers (possibly offshore) is often the most viable option for software businesses. Flexible and effective test process management is especially vital to support the testing life cycle when external testing service providers are involved. Yet, the extant literature provides little theoretical guidance for managing the testing life cycle, including requirements, test, and defect management, and for designing information systems to support the management of the life cycle. The paper of Lu and Käkölä develops an Information Systems Design Product Theory for the class of integrated Requirements, Test and Defect Management Systems based on a literature review and a case study of a Chinese testing service provider. This theory helps geographically distributed clients and testing service providers to manage and control the testing process, make the process transparent and seamless, and improve service effectiveness.

The paper of Leitner et al. describes a lightweight approach to transition from single system development to software product line engineering in the automotive electronic systems domain. This approach supports the two different emerging standards for this domain, changes the existing single system development processes as little as possible, extracts variability information from software models and automatically builds a variability model, and implements a single point of control using the variability model to ensure consistency between various development artifacts throughout the development life cycle and to configure the documentation of a generic software product line architecture. An integrated information system is necessary to support the approach. A prototype of such an information system is evaluated in the paper to partially validate the approach in a laboratory environment.