

# Challenges for the Design of Software-Intensive Systems: Handling Complexity and the Two-Way Alignment Between the Business (Problem) and the Solution (Software)

**Prof. Dr. Klaus Pohl**

Software Systems Engineering  
Director of Institute of Computer Science and Business Computing  
University of Duisburg-Essen, 45117 Essen, Germany  
Email: pohl@sse.uni-essen.de

Software plays an increasing crucial role in success of traditional products such as cars, home entertainment, telecommunication, intelligent buildings. The challenges for the design of embedded software systems is manifold. I will concentrate on two main challenges and their key consequences for design, namely

- a) the *increasing complexity* of software-intensive systems;
- b) the increasing *need to adapt to a fast changing technology and environment*;

The main quality criteria for software-intensive systems is their “fit for purpose”, in both functional and non-functional (better quality) terms. To design systems which fit their purpose, the successful involvement of users, customers, domain experts and other stakeholder throughout the design process is a must. Requirements Engineering plays thus an increasingly important role in the design of complex software-intensive systems. Let me elaborate on the two main challenges stated above in a bit more detail:

**a) *Increasing complexity, cost and design time*:** Reasons for the increase in the complexity of software-intensive systems are, among others,:

- *increase in functionality*: As observable, e.g., in the automotive or telecommunication industry, the functionality of software increases dramatically. Each new product (or even software version) must of course provide all the “old” functionality and, in addition, in-cooperate a lot of new functions – quite often, a platform change is required in parallel, and other devices such as sensors change their behaviour so that the “old” functionality must be adjusted accordingly;
- *increase in quality needs*: The time where the delivery of faulty software or software requiring a significant, not intended adaptation of its environment still satisfied the customer is over. Today, software – as all other products – must be delivered without failure and must fit in the environment. In addition, there are increasing demands for the quality of the internal software “structure”; one main reason for demanding an increase in software quality. Another main reason is the increase of quality features like security, safety or mobility (which often are used to express opaque functional features);
- *mass customisation of software*: Customer demand an adaptation of the software-intensive product to their needs – like with other products like cars. This leads to a mass customization of software-intensive products which quite easily lead to the existence of several hundreds or even thousands of software systems/versions. Maintaining and managing their evolution is a main reason for the increase of complexity – orthogonal to the above.

We thus need techniques and methods which enable software-intensive system providers to deal with this increase in complexity in a systematic way. The principles underlying ***software product lines*** (or families) provide means for those challenges – despite the fact that those principles are by far not fully understood (lets say 10 %). The main difference of software product lines to traditional software design and development are

- the *constructive and pro-active reuse* of software artefacts at all levels – thereby achieving a reduction in development time and costs and at the same time an increase of the quality (through the reuse of quality-assured software assets);

- the *differentiation between the construction* of (a set of ) the system (called domain engineering) *and the production* of customer specific applications (called application engineering);
- the *in-cooperation of variability at all levels of software design* – a key enabler for adaptations and mass customisation as well as for the “derivation” of similar products based on a common core;

To fully explore the concept of software product lines, to develop theories, organisational and technical methods, techniques and tool support as well as economic models like production models, value-added chains etc., requires a significant research investments.

In addition, we need requirements engineering methods, techniques and tools which ensure successful customer involvement throughout the system development and life-time, facilitate mass-customisation and provide appropriate abstraction levels and “transition” mechanism and thereby ease the the complexity.

**b) Increasing need to adaptation:** Business organisations must increasingly change their products and processes. They must quickly respond to new technologies and the business opportunities generated by such technologies. This is equally true for embedded systems, distributed and open systems and for cooperative information systems. Requirements Engineering (RE) is a key ingredient in the design of a global and common software architecture. It is also the key for keeping the architecture aligned and synchronized with the business. Traditionally the alignment between business needs and software solutions was by large a top-down process.

Traditionally, software-intensive product providers faced the problem of “mapping” their business to software solutions (cf. (a) in Fig. 2).

With customer goals evolving rapidly and requiring personalized solutions, this linkage must be tighter (cf. (b) in Fig. 2). In addition, provider must exploit software-supported innovation (cf. (c) in Fig. 2) which is itself increasingly driven by the software solutions available from third-party vendors and communities – in particular the open source and component communities (cf. (d) in Fig. 2).

As a consequence, software-intensive product and service providers, especially from the secondary industry, must deal with a two-way-alignment between the business (problem) and the software (solution). Challenges for design to deal with the two way alignment - all of them requiring significant research investments - include:

- requirements engineering methods, techniques and tools for facilitating the two way alignment establishing requirements as a “bridge” between the problem and the solution space.
- dynamic adaptation and run-time evolution as well as run-time evaluation of software systems;
- quality assessment of components (open source and COTS) during runtime, certification of such components;
- standardisation of components and semantic component interfaces to allow semantic run-time evaluations; even more important as basis for standardisation (or maybe there is another way for construction software systems without having standardised “parts” to be reused during construction? );
- ...

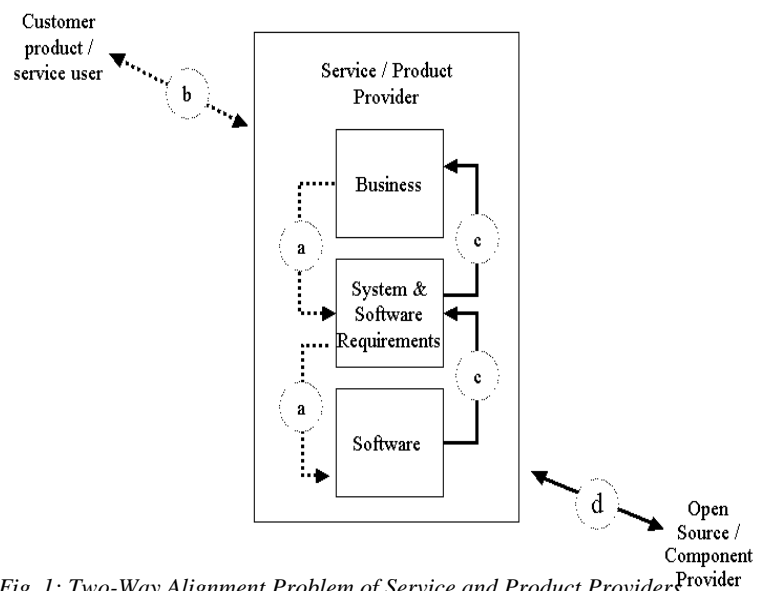


Fig. 1: Two-Way Alignment Problem of Service and Product Providers.