Obviously, software development is a form of modeling—after all, source code is ultimately just a model of a world view. Modeling requires abstraction, i.e., the omission of details. This omission does not just take place at a purely syntactic level. Much more important (and more difficult) is omission at the functional level, i.e., the decision on which section of reality the software should represent. The decision to omit certain functionality requires an understanding of the benefits that are expected from the software to be created. And it also requires a great deal of courage—because the nature of modeling and the uncertainty of software development mean that occasionally, the wrong things will be omitted. This courage is essential though, because including all conceivable ideas in the model to stay on the safe side (and implementing them in the product) leads to bloated software that is expensive to create and expensive to maintain.

Making reasonable decisions on a software’s scope is only possible with domain knowledge. For example, let us assume that we have no idea of what the examination system at a university looks like. It seems extremely unlikely that we would be able to provide the correct responses to the question of what is required in an examination information system to be used by examination offices, lawyers, and university lecturers. In other words, a distinction between what is important and what is unimportant requires knowledge of the application domain.

This is where the Interaction Room comes into play. It brings all people responsible for developing the software product into the discussion. This most likely includes developers and future users as well as technology and business experts. They must develop a common idea of what the software to be developed must provide, what is essential and what is expendable, what is possible using the technology, and what should be left out. All this is necessary to ensure that the software is not just correctly developed (developers with a certain amount of experience can generally do this alone), but to ensure that the correct software is developed—software that provides the greatest possible benefits given the available resources. To evaluate this, developers must understand the objectives and priorities of future users. To achieve this, the Interaction Room (IR) does not follow the
myths and rituals that are often seen in software engineering, especially those that provide the illusion of completeness and consistency.

The Interaction Room is a physical room whose walls are covered with models of business processes, business and physical objects, as well as user journeys and system landscapes. It is a room in which communication is encouraged and facilitated, and whose finite walls make it clear that the focus must remain on what is important. It is a room which makes it obvious that a business data model should better contain 40 rather than 140 object types, and that 15 core business processes can describe the purpose of the system more clearly than 50 special cases.

The work in the Interaction Room does not follow a completely closed methodology (in the sense of a number of steps that lead from a problem to a solution in a certain order). Rather, the following chapters describe a range of method fragments that can be combined in different ways in different project situations.\footnote{However, for the sake of practicality, we will continue to refer to the sum of the individual method fragments as the Interaction Room method.}

An Interaction Room promotes moderated and targeted communication between a project’s stakeholders, focuses on what is important and ensures that required features are evaluated and prioritized in light of the desired added value. The latter occurs using annotations, which allow every stakeholder to express their ideas of the key objectives and features of the desired solution. An Interaction Room supports the scoping of projects as well as the pursuit of project progress at a qualitative and quantitative level. It creates transparency and allows stakeholders to jointly coordinate the direction of projects, respond to risks and changing expectations, and continuously work toward creating a lean software solution.

### 2.1 Key Interaction Room Principles

The Interaction Room ensures that the key principles of every project, namely abstraction, value orientation, communication, and transparency, do not just remain empty words, but instead become visible and tangible:

- The principle of abstraction demands a focus on the key relationships and genuinely essential decisions. The aim is to leave out details at certain levels of abstraction, while remaining aware that they will have to be filled in at a later date and that these details may subsequently play an extremely important role. Which details may be omitted, and where, is the subject of agreements, methodology, pragmatism, and common sense. In particular, models overloaded with details are more dangerous than incomplete models, especially in the early stages.

  In the Interaction Room, the abstraction requirement is manifested in the finiteness of the walls available for model sketches. It ensures that every