

# Chapter 1

## Introduction

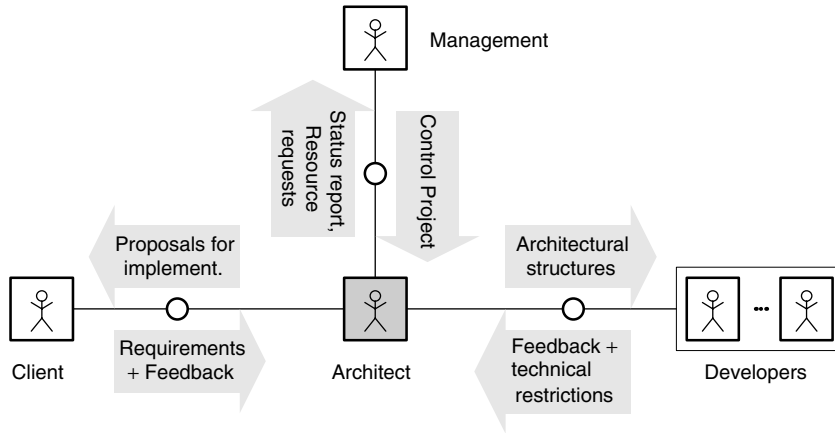
“But the business of software building isn’t really high-tech at all. It’s most of all a business of *talking to each other and writing things down*. Those who were making major contributions to the field were more likely to be its best communicators than its best technicians.” – Tom DeMarco [DeM95]

### 1.1 The need for communication

#### 1.1.1 Software and people

Developing software is a hard task. Many approaches have been followed in order to implement more functionality in less code statements and to support coding with powerful tools. Unfortunately, software projects are still hard to handle. One reason is that the ‘human factor’ becomes more and more important, as:

- software is being developed in teams;
- many projects require an integration of third-party products or other existing software which have to be understood; and
- coding only takes a small fraction of the total time to develop a software product, as collecting requirements, organizing teamwork and discussing concepts becomes more and more important.



**Figure 1.1:** Stakeholders have to communicate

In a nutshell: being able to understand, describe, and communicate technical information becomes an increasingly important skill required by architects, developers and project managers.

Figure 1.1 shows a typical project set-up from the communication point of view. The architect communicates with the client, the management and the team of developers. For each of them, he or she has to focus on different aspects and use different levels of technical detail.

This chapter outlines the communication tasks of an architect which will be addressed in this book.

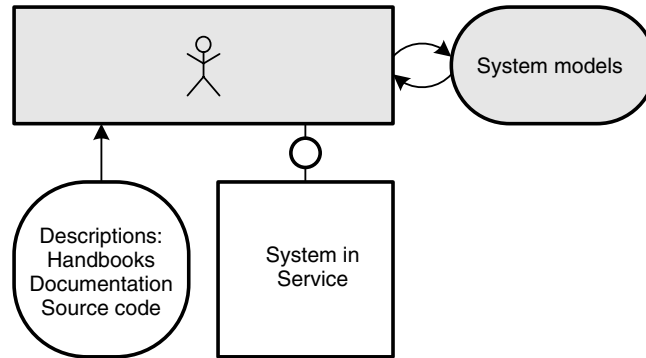
### 1.1.2 Structure information to prepare communication

“Take a look at the product XY and tell us if we can build on it in our project.”

There are so many systems, platforms and technologies that you have to understand before you can write a single line of code. By learning a technology, or analyzing an existing system, you need to build a model of it in your mind.

#### Modeling

A *model* is an abstraction of an existing or planned system which comprises only those aspects which are relevant to its purpose. A system model can be used, for example, to communicate, construct or analyze.



**Figure 1.2:** Structure information about a system

*Modeling* describes the process of creating a model, which includes choosing the most important facts and leaving out others. This *abstraction* heavily depends on the purpose of the model and its addressees.

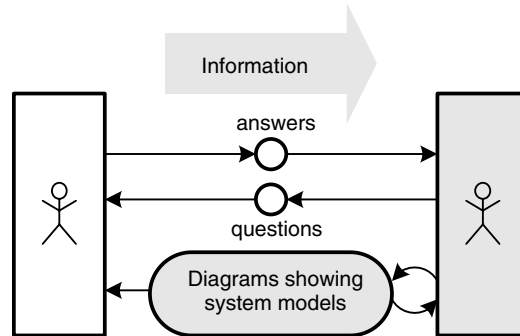
By jotting down the models, you are able to recall this knowledge later on. Furthermore, you can use them to communicate this information to other people.

There are many modeling techniques and notations which address various purposes. This book will present the Fundamental Modeling Concepts (FMC) which were developed to support communication about technical systems.

The FMC block diagram in Figure 1.2 shows a person who reads system descriptions, such as handbooks, documentation or the source code, observes and uses the system in service, to work out a new set of system models. In this case, the architect or developer uses modeling to structure information about a system, to abstract from too many details, or to obtain an overview.

### FMC Block Diagrams

A *block diagram* shows the compositional structure of a system. The building blocks are *agents* (rectangular nodes) which process information, and *locations* (round nodes) which are used to store or transport information. Lines or arrows connect agents and locations and depict which agent can access which location, including the direction of information flow. Block diagrams are introduced in Chapter 2. A notation summary can be found on page 300.



**Figure 1.3:** Retrieve system information from people

### 1.1.3 Retrieve system information from people

“You must have missed this topic in the list of requirements!”

“Ask someone from department X to explain their module, we need to integrate it into our product. Unfortunately, they are very busy at the moment.”

It is not unusual that important system knowledge can only be found in the heads of their developers and architects. Even worse, sometimes it is spread over many people but no one has a complete picture.

This scenario is similar to the previous Section 1.1.2, but in this case you need to interview humans to obtain technical information. Many factors can affect efficient communication. Even assuming that all interviewees are willing to share their knowledge, there is still the danger of talking at cross-purposes or of focusing on unimportant details.

You can support interviews with diagrams, as shown in Figure 1.3, because you can show the interviewee what you have understood so far, and establish a common terminology. Furthermore, working with graphics to grasp abstract structures makes communication more effective – there is something to ‘point at’ while discussing.

#### **Iterative Modeling**

Instead of creating an accurate model after you have collected all the facts, it is often better to start with an incomplete or potentially wrong model which shows what you have understood so far, and let the knowledge bearers correct you. Prepare diagrams which serve as starting points for subsequent interviews.

The same applies to creating a system architecture. Allow for iterations of the system model, give every team member the opportunity to consider their aspects and the consequences for the architecture.

Another scenario also deals with obtaining information from people: finding out the customer's requirements. A typical problem of requirements engineering are requirements which come up after the development has already begun. It is simply not realistic for a customer to think of every requirement before he or she has seen or even touched a solution, usually a prototype.

Mental Prototypes can support the process of finding and formulating requirements. A *mental prototype* is a model of a system which outlines a potential system solution. It describes functionality, but leaves open most of the implementation details. The customers can imagine how the system works, check if their requirements are met and whether they are happy with this solution.

Mental prototyping only works if customers can understand the model easily and can execute the model in their mind.

#### **Mental Prototype**

System model showing an abstract solution to the customer's requirements. Serves to get feedback from the customer and to refine or discuss requirements. See Section 8.3.1.

### **1.1.4 Communicate system information to people**

"I will write this program for the customer, but please don't ask me to explain it to them."

"Don't annoy me with those marketing diagrams. I want more substantial information, but no source code, please!"

"Can you please explain . . ."

Sharing knowledge is a crucial task when working in a team. Communication becomes more complex if you have to address more than one or two people, and if they have different knowledge and expectations.

In this scenario, you need to consider didactical issues, as you need to transport your knowledge to other people in an efficient way. As Figure 1.4 shows, diagrams of system models can support this process, assuming that the addressees understand them quickly.

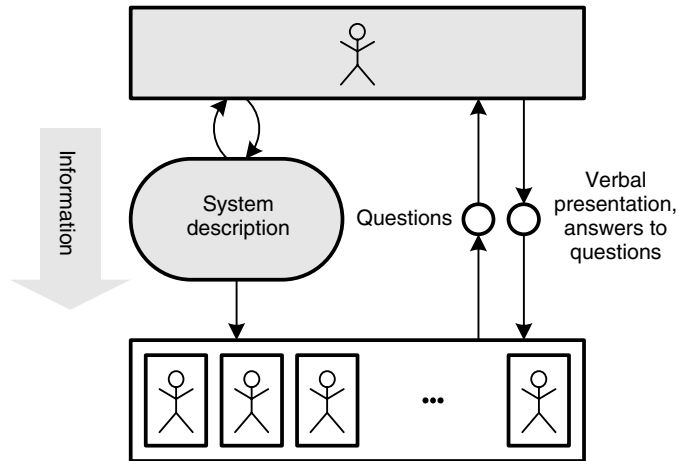


Figure 1.4: Communicate information about a system

### Didactical Modeling

Use diagrams of the system model to transport information to other people. Focus on these diagrams while explaining the system. Get feedback from the attendees about comprehensibility, apply their suggestions to your final report. See Section 8.4.2.

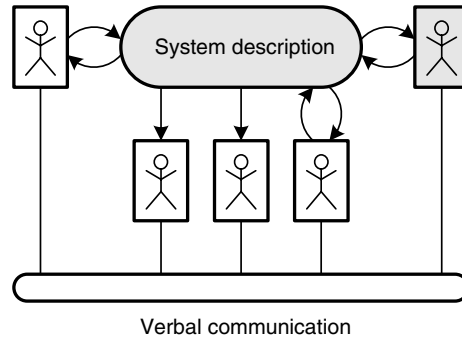
## 1.1.5 Support communication

“I don’t think we’re talking about the same subject.”

It is not unusual for people to talk at cross-purposes, so use the same words even though they may have a different meaning to them, or do not contain the same level of detail.

If the discussion is about technical issues, you can avoid this situation by using system models to focus the discussion (see Figure 1.5). Use pencil and paper or the whiteboard to point out what you have understood so far and ask if everyone agrees with that. The discussion will usually focus on your diagrams. Invite participants to modify them to show their point of view. Section 8.4.1 introduces some steps to support meetings.

Figure 1.5 shows five people communicating via a common channel at the bottom. They also use system models to support communication (three modify them, two only read them).



**Figure 1.5:** Support communication about a system

### Ad Hoc Modeling

Modeling a system which is currently discussed using no other tool than a pen and a whiteboard or paper to support communication. Ad hoc modeling serves to obtain agreement on a common terminology and a common level of detail and helps to focus the discussion.

### 1.1.6 Structure teamwork

“I need to know who’s working on which topic.”

“You too? I thought it was my task to design this interface!”

“My colleague is ill and I have no clue what he’s been doing for the last months.”

Managing a team of developers requires a great deal of communication. The project manager has to know who is working on which part of the system and has to make sure that the parts will later work together. It is therefore useful if the project manager can map the tasks of each developer to the elements of the system architecture whenever possible.

The architecture of the system to be developed should be known to everyone who is working on it. Developers should know the purpose of the components on which they are working, their future interaction with other components and the names of those who work on them.

A good means of mapping tasks to architectural elements is the *System Map*. It is a diagram showing a detailed model of the system and its architecture. After being developed by architects, it serves as a basis for discussion about details and should be known by all team members.

### System Map

A Model showing the complete system at a reasonable level of detail. It depicts what every team member must know about the system and allows them to locate their work and their responsibility. See Section 8.3.2.

## 1.2 The FMC Idea

### 1.2.1 Requirements for a modeling technique supporting communication

Using system models to support communication among people sets some requirements for the modeling technique and its notation:

- **Abstraction.** The ability to describe the conceptual architecture on many different levels of abstraction is of major importance.
- **Simplicity.** A description technique should be restricted to a few elementary concepts and notational elements.
- **Universality.** Although being simple, a description technique should also offer enough expressive power to cover a wide range of system types without being bound to a specific paradigm.
- **Separation of concerns.** The description of complex systems has to include different conceptual aspects. It is important for an architectural description technique to support the separation of these aspects by offering comprehensive means for their illustration.
- **Aesthetics and secondary notation.** The notational elements of an architectural description technique should support proper layout including the easy formation of graphical patterns.

### 1.2.2 FMC

The Fundamental Modeling Concepts (FMC) presented in this book are examples of a modeling technique created especially to support the communication about information processing systems. FMC uses a simple notation which can be used easily for ad-hoc creation of models in meetings. FMC is not tied to a programming paradigm, and can even be used for information processing systems which are implemented with a hardware / software mix.

FMC has been created and refined by Siegfried Wendt and his group in many industrial projects. FMC is based on established concepts such as Petri nets and Entity / Relationship modeling, and has been consequently tailored to support communication.

### 1.2.3 Three aspects – Three diagram types

#### Three fundamental aspects

FMC separates three fundamental aspects of an information processing system:

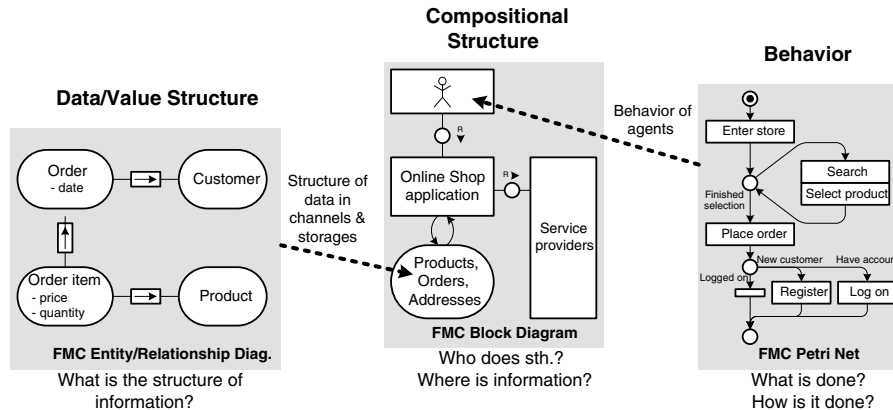
- *Compositional Structure* describes the structure of the system in terms of active and passive components. Agents are active and process information, while storages and channels are passive components which keep or transport information. With the help of this structure, we can see which agent can access which storages and communicate with other agents via channels or storages.
- *Behavior* of agents that operate on data, and respond to requests which they receive via channels.
- *Data / value structure* describe the structure of information which can be found in storages or is sent via channels, as well as the relationships between data in different storages.

#### One diagram type per aspect

For each of the three fundamental aspects, FMC provides one diagram type, as depicted in Figure 1.6 which shows block diagrams for the compositional structure, Petri Nets for the behavior, and Entity / Relationship diagrams (E/R diagrams) for data / value structure. This figure also shows which questions are answered by which diagram type.

#### Relationship between the diagram types

The compositional structure provided by a block diagram is the cornerstone of an FMC model. Petri nets then describe the behavior of an agent or a set of agents that can be found in the block diagram. Entity/Relationship diagrams are needed if the structure of data and their relationships require a more abstract description. Nevertheless, this data can be located in storages and channels of



**Figure 1.6:** Three aspects of an FMC model and their corresponding diagram types

the block diagram. Figure 1.6 shows the relationship between the three FMC diagram types.

Take a look at Section 6.1 to learn more about the FMC meta model describing the elements of the diagram types and their relationships.

### 1.3 Outline of this book

Chapters 2 to 5 deal with the introduction of the three FMC diagram types: Block diagrams for compositional structures, FMC Petri Nets for dynamic structures, and Entity/Relationship diagrams for value structures. These chapters also provide exercises to help you recapitulate what you have learned so far.

Chapter 6 goes into greater detail of FMC modeling. It introduces the FMC meta model and advanced modeling concepts, such as structure variance, abstraction and refinement, or non-hierarchical transformations.

As most information processing systems are implemented using software, mapping the system structures of FMC models to implementation is an important aspect which is discussed in Chapter 7.

Chapter 8 focuses on ways of applying FMC in your daily work. The scenarios described there originate from the authors' experience, but the list is not exhaustive. Most of them deal with communication.

The intention of FMC modeling is to support communication. This makes high demands on comprehensibility of diagrams, which depends on various factors that are the subject of Chapter 9.

Chapter 10 discusses FMC's relationship to other modeling approaches such as Structured Analysis (SA) and the Unified Modeling Language (UML).

The pattern language for request processing servers in Chapter 11 is one of the results of the analysis of various server systems and shows how FMC diagrams can be used to describe server concepts in a compact and concise way.

The appendix of this book provides, among other things, reference sheets for the notation of the three FMC diagram types, and a glossary of important FMC terms.

