Chapter 2
Software Products: Terms and Characteristics

Software is an intangible economic good, with no physical form, its utility or value not even perceptible in another form. So only the functionality of software is perceptible e.g. via a user interface, or as the result of a controlled transaction via software, e.g. as an account movement. What exactly constitutes a software product is often rather subjective. As with many highly technical products, many people do not understand how software products work. Software is therefore in the truest sense of the word "intangible." Software thus contrasts greatly with other investments or acquisitions of consumer goods. In particular the customer does not really acquire the product when buying software, rather very specific, precisely defined rights of use in a license contract. However, investments in software today represent a larger proportion of spending for IT infrastructure than investments in hardware, and software contracts with large companies often amount to multiple million of dollars.

Software belongs not to the three classic economic factors of capital, land, and labor, but in the new fourth category of "knowledge." Software is the manifestation of human know-how in bits and bytes and in this form also possesses the invaluable advantage (and simultaneous disadvantage), that it can be easily copied and quickly circulated over any distance. The right software, ideally applied, can represent a more important strategic competitive advantage in today's economic life than all the other factors. Software can be crucial for competitiveness in production processes, functionality, the availability of service products, and thus for a company's success or failure on the market. But what is a software product and what is it not – or not yet? What role does the price play? How should we classify services that are offered on the basis of software?

We find it reasonable not to limit the term "software product" to the world of software vendors, but also to use it in the world of corporate IT organizations. Is, for example, an online account with a direct bank, a mobile phone in combination with a special mobile phone tariff or the membership to a chat community a software product? In this chapter we attempt to define the term "software product" and to discuss certain features of software and their relevance for products and product management.

Marketing generally defines the term "product" as follows: "a product is anything that can be offered to a market for attention, acquisition, or consumption that might satisfy a want or a need." (See [KotArm07]).

H.-B. Kittlaus and P.N. Clough, *Software Product Management and Pricing.*

In our definition we avoid the market term (which focuses the conceptualization too much on a "mass market" for our purposes) and prefer to put the relationship between two parties in the foreground, so that individual developments and internal customer/supplier relationships can be included.

Product = Combination of (material and/or intangible) goods and services, which one party (called vendor) combines in support of their commercial interests, to transfer defined rights to a second party (called customer).
Software product = Product whose primary component is software.

The word "party" indicates that we are not necessarily dealing with a corporate entity. It can also be areas within a company or an individual person. The phrase "in support of their commercial interests" should make clear that it refers to business but does not necessarily lead to a payment. There is also a commercial interest behind Open Source even if only to harm the established software vendors. Even a product free of charge (e.g. the Microsoft PowerPoint Viewer) has a commercial goal – to increase market penetration of another product from the same software vendor which is subject to a fee. The phrase "defined rights" expresses that there is some room for variation here, e.g. right of use (possibly with restrictions), property right, right to resale etc. Details are typically defined in the software vendor's licensing terms or in an individual contract between the parties concerned.

This product definition should also express that something can already be a product when it has not yet been bought by a customer; that is it does not become a product through the buying process, but through the intention to sell it. In establishing the boundaries of what a software product is and is not, we have knowingly chosen a "flexible" phrase: the word "primary" should make it clear that there is room for discretion.

A mobile phone is not a software product according to our definition (rather a telecommunication product), even if software is an important part and may have absorbed a large proportion of the development costs. In this case we would be talking about embedded software, which "serves" the function of making phone calls and is therefore an underlying part of the whole product, which cannot be bought separately. We define:

Embedded software = A piece of software that is not sold as a stand-alone software product, but is integrated in a non-software product.

Embedded software does not manage and operate only a computer or a processor, but rather is embedded in a technical system which consists of other components as well, which allow the whole thing to become a product. This can be the software for programming and managing a machine, but also diagnosis software for finding errors in an automobile or software for servicing a dialysis machine in medicine. These programs serve highly specialized interfaces, which are typically very closely integrated with hardware. Therefore the requirements and the product management are driven by the functionality of the complete system. According to our definition from the beginning of this chapter, since embedded software is not a stand-alone software product, it will not be discussed further in this book.

Another important term in this context is "OEM product". OEM stands for Original Equipment Manufacturer. The term was originally coined in the hardware business and later transferred into the software world. It means that one manufacturer sells one of his products to another manufacturer who uses it as a component in one of his products without showing its origin openly. We define:

OEM software product = software product of software vendor A that is used by company B as a component under the covers of one of B's products.

Notice that B's product can be, but does not have to be a software product. A vendor is usually willing to sell his products as OEM products at a significantly reduced price in order to increase the volume. We will discuss the pro's and con's of this approach in chapter 5.

Let's look at some more examples. A games console is also not a software product (rather a games product). The same arguments are valid here as for the mobile phone. A game for this console – purchased separately, separately packaged with its own price and own terms of licensing – is obviously a software product, however.

An online account is not a software product according to this definition, rather it is a banking product realized with help of software (products) within the bank. The customer receives online access to his account from his bank, which allows him to carry out bank transactions (bank balance enquiries, transfers, establishment of standing orders etc.) at home or wherever he happens to be. The software is only useful and usable in connection with the account. So the account is the primary component.

A search offering like Google qualifies as a software product even though that may be contrary to some people's intuitive understanding. It does fulfill all the criteria of our definition: there is a commercial interest, the customer gets the right to use it, and its primary component is software. This approach is close to the "Software as a Service" model (SaaS) that we will discuss in more detail. Google's case is special since the price is zero, but Google makes significant profit with the advertisements displayed with the search results.

2.1 External and Internal Views on a (Software) Product

By changing the perspective or the point of view, a "non-software product" can very quickly become a "pure" software product. Let us keep the example of the online account from the last chapter: The online account is seen as a bank product through the eyes of the bank's end customer or the bank's executive board. For this purpose, there may be a product manager in the bank's organization, who works on the customers' requirements and generally makes sure that the bank has a consistent and competitive home banking offer for its customers.

The online account is realized internally via several hardware, software and service components, which can originate from various sources. Let us accept that the product is realized by the components online application, HBCI (Home Banking Computer Interface) server and a call center. Furthermore:

- The online application is created by Application Development as part of the corporate IT organization.
- The HBCI server is bought as standard software and integrated with the online applications by the corporate IT organization, tested and put into production.
- The call center is outsourced to an external service provider.

From the point of view of the corporate IT organization, the "online application" component is a software product. Since it was developed internally, an internal software product management is needed that receives – at least functional requirements – from the product manager of the bank product "online account". Furthermore, the bank may decide to sell this software product to other banks, which would underline the necessity of an explicit software product management.

For the standard product "HBCI Server" on the other hand, the bank is a customer of a software vendor. Since the product does not fulfill all of the bank's specific requirements from the perspective of the IT organization, corresponding requirements are given to the product manager of the software vendor who has to commit to their implementation.

The call center, including infrastructure and agents, was completely handed over to an external service provider. Service level agreements (e.g. average and maximum time that the customer is supposed to spend in the queue) were negotiated and secured contractually; no requirements were placed on individual software products.

The external service provider implements the call center for the bank via hardware and software as well as personnel recruitment. This includes components such as ACD (Automatic Call Distribution), CTI (Computer Telephone Integration), and IVR (Interactive Voice Response) Units, call recording etc. As the service provider offers call center solutions for many firms, this is one of their main products from their point of view, and therefore they assign a product manager who manages requirements from customers (e.g. the bank) and elicits requirements against the integrated hardware and software products from other vendors.

As this example shows, which can easily be detailed even further, the definition of what a product is depends on the individual perspective. Many non-software products have software products inside. So the topic of software product management often plays a role in those product areas as well.

2.2 The Software Product as Type and the Customer-Specific Installation as Instance

To understand the term "software product," we must differentiate between type and instance. For example, the development of an automobile, e.g. of the Audi A4, defines a type. This can have a multitude of different parameters which influence the production process.

Audi A4 type
Model: *Sedan, Avant, Cabriolet*
Engine: *2.0, 3.2*
Color: *grey, white, red, beige, blue*

When ordering a specific car for a customer, the model features are specified (product configuration). In this way, during the manufacturing process an instance of the A4 type is created.

A4 for Fred Miller = Audi A4
(Model: *Sedan,*
Engine: *3.2,*
Color: *red)*

Afterwards the instance is delivered to the customer and no longer changed by the manufacturer or the customer.

The product manager for the Audi A4 is therefore responsible for the whole "type" of A4; even when there are many different models, the number is always finite and all combinations are known before production begins.

Software products can also be distributed by the manufacturer in diverse variations. However, in many cases, not only a selection of a variation before distribution is carried out, but a customer-specific customization of the software product to its environment on location. This can be carried out by the customers themselves, or with the support and advice of the software manufacturer or a consulting company.

Salary and wages type
Platform: *IBM AIX, SUN Solaris, HP-UX, Linux*
Language: *German, English, French, Italian*
Tax table: *Germany_2009, Italy_2009, Switzerland_2009, Austria_2009, England_2009, France_2009*
Medium: *CD, Download*

S&W for Vienna = Salary and wages
(Platform: *SUN Solaris,*
Language: *German,*
Tax table: *Austria_2009,*
Medium: *CD)*

If parts of the product are also distributed in source code, the product itself can be changed beyond pure customization. For this there are unpredictable numbers of possibilities, which cannot all be considered, planned, and checked by the product manager. Nor does the software vendor wish to be dependent on code for which he does not control the quality. For all these reasons, as a rule the manufacturer's guarantee relates only to items originally distributed by the vendor, but does not apply to code modified by the customer.

2.3 Product Platform, Family, and Line

For technology companies it can make sense to differentiate between product platform, product family, and individual product. McGrath defines ([McGrat01]): "A product platform is not a product. It is a collection of the common elements, especially the underlying defining technology, implemented across a range of products. A product platform is primarily a definition for planning, decision making, and strategic thinking. The choice of a defining technology in platform strategy is perhaps the most critical strategic decision that a high-technology company makes." So we define:

> **Product Platform** = the technical foundation on which several software products are based.

An example of a product platform is the SAP core system that serves as the basis for all SAP components. McGrath sees the product platform as key competitive factor the management of which must be a core competency of a company. For him a product platform must be controlled by the company. For software there can be platforms that are not under control of the software vendor but which nevertheless may have key influence on the success of their product. For example, for any kind of PC software the question of the operating system platform(s) needs to be answered. For some time the decision for Microsoft Windows was easy because of its dominance in the market whereas several years ago IBM's OS/2 was a valid option, and today Linux makes the decision more difficult again.

A product platform is usually not an independent product, but rather a combination of technological elements used in various products. Such a platform often constitutes an especially valuable asset and serves as a market differentiation factor. It therefore requires very sensitive management, since errors will immediately have serious consequences for all products based on the platform and thus for the company as a whole. An example is Wang Laboratories, which dominated the word processing market in the late 70s with its combined hardware and software products. Wang viewed the combination of hardware and software as a defining element of its product platform and continued to do so – which proved to be a fatal mistake – even after the computer market had developed and customers thus no longer wanted hardware intended solely for word processing. If Wang had, at the time, realized that the defining element of its product platform was software and ported this software onto the new computer platforms, Wang might still be the major supplier of word processing software today. In fact, Wang disappeared from the market, giving Microsoft the opportunity to take over the word processing market.

A positive example on the part of commercial users is Amazon, the large Internet e-commerce company. Amazon developed a product platform and defining technology comprised of software that today not only sets the standard for e-commerce, but also serves as a basis for geographic and product-range-related expansion of commercial transactions.

A platform that brings real competitive advantage may serve as a base not only for one product, but for a family of products. The establishment of a product

family in the market, however, is motivated by pure marketing reasons. An example is IBM's DB2 family. In the 1990s, IBM combined all relational database products on the various system platforms to the DB2 product family and gave all products this name (which had previously only been used for the host product). However, this was not based on a joint code base of the individual products, even though this would have been preferable from a development perspective. In any case, customers associated the same family name with a large degree of similarity. We define:

> **Product Family** = A group of software products which for marketing reasons are marketed as belonging together under a common family name.

A software vendor groups various products together under a "family name" which can then be marketed more efficiently than a single product. This approach suggests that the products belong together, implying that they are either technologically similar or that together they provide a solution for specific problems. The technological similarity can be a common product platform, e.g. SAP products, or a common basic technology, e.g. IBM's DB2 family. Microsoft Office is an example of a product family comprising a group of products that address specific problems, in this case office tasks, even if the components of Office have not always had a seamless relationship. The above examples illustrate that the terms "product platform" and "product family" sometimes coincide, i.e., products based on a common product platform can be – but do not have to be – marketed as a product family. Conversely, products that are marketed as a family can have – but do not have to have – a common product platform. Whether or not it is advisable to establish a family concept is primarily a marketing decision, which does, however, have an impact on the requirements for the products concerned. Customers expect products belonging to a product family to exhibit more common features in terms of integration of product combinations or interface similarities in the case of technologically similar products. A negative example is the Norton security products System Suite and Utilities, which are fundamentally one menu from which multiple unrelated programs may be invoked. If the products do not adequately meet customer integration expectations, the family concept can have a negative market impact.

The term "product line" has gained a lot of attention in the last couple of years, primarily in, but not restricted to the area of embedded software. We define:

> **Product Line** = A group of software products which are variants of a base product governed by a common software architecture.

An example is an application software for offer calculation for craftsmen that needs to be adapted to any specialized craft it supports, like plumber, gardener, etc. Here adaptation means more than just customization, i.e. it means not just setting parameters differently, but requires changes to parts of the code of the base product. That is why in this case we do not call the base product a product platform which by definition would have to be identical for all products based on it.

All three concepts, product platform, product family and product line generally increase the complexity of software product management (see chapter 4).

2.4 Product Name, Version Numbers and Compatibility

For most software products development continues throughout its commercial (sales) lifetime. An important question in this context is if and when the development of a product results in a different product (with a new product name etc.) or simply a "new edition" of the existing product. There is no universally applicable answer to this question. Marketing aspects frequently play a more important role than technical aspects when selecting a name. It has become common to denominate software versions after a specific nomenclature, which is generally dependant on the manufacturer, however.

IBM uses a three-level hierarchy of software levels, for example:

- **Version:** Denotes a new product, which as a rule comprises crucial expansions and improvements. Furthermore, a new version is always subject to a fee and also always receives a new (internal) product number.
- **Release:** Denotes a new level of software with bigger functional or other improvements. New releases are generally free of charge for customers with a maintenance contract, and keep the product name, product number and the price model from the predecessor.
- **Modification Level:** Denotes a new status of software with limited expansions and delivery of error and cosmetic corrections to the predecessor.

An example of complete product identification is IBM z/OS Version 1 Release 9 Modification Level 5 (in short z/OS 1.9.5).

Anecdotally, customers seem to be more accepting of new version designations (a.k.a. new product) if the version is declared on an obvious boundary and the most satisfactory boundary seems to be a machine architecture and operating system change boundary.

There may be a number of motivations to rename a follow-on product. If, for example, the product currently on the market had a quality problem during introduction this may have led to an image problem in the long term, which is naturally associated with the product name. In this case, a new name may signal a new beginning and as little in common as possible with the previous product. A new name can also be necessary due to the standardization of product names in connection with a larger product portfolio or a product family like IBM's DB2 family (see above).

Frequently, further development is needed – if not mandatory – due to technological progress and changes in products which serve as a basis for the software product (prerequisites). For example, an operating system must support new hardware (processors, storage mediums, etc.). The old version of the operating system will be taken out of service at some point, so that – often with a certain time delay – a chain of upgrades is required for the system and application software products on higher layers of the technology stack. In this case, the version and release numbers underline upward compatibility of products and investment

protection ensured by the vendor for its customers. Therefore, a change in the product name should always be well considered to prevent potential irritation of the customers. Naturally not all software vendors adhere to this "unwritten" law of upward compatibility, and there are enough examples in which the product name was kept and the customers experienced a nasty surprise when changing to the new version.

By upward compatibility, it is understood that:

- In changing from software version n of a product to the next version n+1, existing functions of version n continue to be supported.
- Data from version n can be transferred to and used with version n+1 without changes.
- Interfaces of version n (APIs, Interfaces for other Systems/Products) remain unchanged.

Should only parts of these conditions be fulfilled, we speak of function, data and interface compatibility.

Frequently, with a new release of a product comes an expansion and change of the underlying data model which leads to changes to the data structures. In this case, data compatibility cannot be achieved easily. A separate data migration is required, for which the software vendor should preferably provide in the form of procedures and scripts.

By downward compatibility, it is understood that:

- Data from version n+1 can be transferred to and used with version n without changes.
- Version n+1 can communicate to Version n, i.e. Version n interfaces are supported.

In contrast to the upward compatibility, downward compatibility cannot always be expected or presumed. For example, a document created with MS Word 2007, as a rule, cannot be read using an older version of MS Word and must be converted before being read to the internal format of the older version. The opposite way should not lead to any problems. (Note: in response to customer demand, Microsoft has made available add-ons to some of their old Office products which allow opening the newer file types. From personal experience we can assert, however, that there remain incompatibilities, e.g. Formatting in Excel 2007 vs. Excel 2002).

2.5 Attributes of Software Products

Software products have specific attributes, which classify software in more detail and which help to highlight certain aspects of product management, as different kinds of software products pose different questions and also demand different approaches in product management. Basic description criteria are, for example:

- Market:
  - Consumer (B2C), e.g. games software
  - Business (B2B)
    - Horizontal (i.e. across many industries): e.g. systems software, middleware
    - Vertical (i.e. industry specific): e.g. securities application for brokerage industry
- Functional areas, e.g. systems software, middleware, application
- Development focus, i.e. standard software vs. services vs. individual development
- Conditions:
  - Terms of contract, e.g. open source, freeware, shareware, priced licensing, SaaS
  - Development at a fixed price or a price according to effort

The term "services" has emerged as a new topic as the base of service-oriented software architecture, often in the form of web services. Web services are usually offered via APIs (application programming interfaces) that can be accessed over a network, such as the Internet, and executed on a remote system hosting the software that performs the requested services. The business models and marketing and operation strategies for services are currently still in the experimentation stages, even though the concept has been around for quite a while.

In every software installation, there are three basic types of software products: 1) the operating system, 2) middleware, and 3) applications. The operating system provides a standard base of system facilities (e.g. the ability to read and write to storage) and takes care of interactions with the hardware. The term "middleware" refers to the software that connects software components and sits "in the middle" between the operating system and the applications. It includes web servers, application servers, database systems and transaction monitors. Finally applications are written to perform functions requested by end users, so that they can accomplish useful business or personal tasks with their computers.

An especially important differentiator of products is triggered by market criteria, i.e. the question, is the customer a company (Business-to-Business or B2B) or a consumer, i.e. packaged software products for the mass market, which appeal to individual end users (Business-to-Consumer or B2C). Mostly, this refers to PC or games software.

Of course, there are overlaps between B2B and B2C. The best examples are PC operating systems, security products, and office products. Usually packaging and pricing differ for these target markets. While PC products for the end user market are typically sold at the list price as so-called shrink-wrapped products with all the corresponding media (software on CD, printed documents) or via download on the internet, the same software is offered to a company with a multi-user or enterprise license with just one copy of the media or download rights at an individually negotiated or volume offering price. Furthermore, many vendors, e.g. Microsoft, create variations of their base PC software products by limiting functionality. Thus they can sell it at different price points in the end user market.

There are clear differences between these two product areas. Business software, in comparison to consumer software must be customizable for the specific needs of the company. The expenditure for adaptation and implementation of software products for business customers is typically in the same range as the price paid for the software or may even exceed it. Studies have shown that, for example, firms who employ ERP (Enterprise Resource Planning) software typically spend 30% of the total expenditure on the licensing of the software products and 70% on services for customization and implementation. Some software vendors perform this work for their customers themselves; others predominantly leave such activities to partner companies in the service sector. The advantages and disadvantages of the respective strategies will be more closely looked at in Chapter 4. In connection with this high investment in customization and implementation, the installation of more complex business software products typically lasts for months, while PC users are used to installing a product within a few minutes and immediately using it in a productive manner.

A further difference between business software and consumer software lies in the fact that consumer software is developed and marketed for millions of individual customers, while the number of customers for business software is clearly lower, but with significantly more complex installations on networked systems with a large number of users.

Business software and consumer software require significantly different prioritization of their most important product management tasks (see [HoRoPL00]). For consumer software, the marketing strategy is at the top of the list of priorities, followed by the partner strategy and the question of whether and how quickly a product will be offered on the international market. For business software, the partner strategy has the highest priority, followed by the service strategy, thus the question with which partners or resources the successful implementation of the product can be guaranteed to the customer. The marketing strategy is the next priority for business software, interestingly only in third place.

The influence of these characteristic attributes on software product management will be discussed in detail in Chapter 4.

http://www.springer.com/978-3-540-76986-6

Software Product Management and Pricing
Key Success Factors for Software Organizations
Kittlaus, H.-B.; Clough, P.N.
2009, IX, 231 p. 17 illus., Hardcover
ISBN: 978-3-540-76986-6