Software Vendor Strategies 3

Against the background of the economic principles described in the previous chapter, this chapter examines selected strategies for software vendors. The vendor's positioning within the value chain is of critical importance. Section 3.1 initially considers the opportunities and challenges associated with cooperation strategies. In this context, we also look at acquisitions, which play a key role in software markets. In addition, we discuss sales strategies (Sect. 3.2) and pricing strategies (Sect. 3.3). We conclude by exploring key management questions concerning the development of software in Sect. 3.4.

3.1 Cooperation and Acquisition Strategies

This section looks at cooperation and acquisition strategies within the software industry. As already described in Sect. 2.2, these strategies are of central importance, in particular against the background of network effects on software markets. Moreover, a single vendor is unlikely to be able to fulfill all customer needs with its own portfolio of products and services. First, we will highlight the general benefits and challenges of cooperation (Sect. 3.1.1). We will then investigate acquisitions (Sect. 3.1.2).

3.1.1 Cooperation in the Software Industry

3.1.1.1 Advantages

Cooperation refers here to collaboration between legally independent entities (based on contractual or tacit agreements). We assume that cooperation is for the medium or long term, and requires capital investment on the part of the participants. The key goals are to secure greater efficiency or value added of a kind that would not have been possible without cooperation. Brandenburger and Nalebuff refer in this context to a "value net" created by this collaboration (Brandenburger and Nalebuff 1996, pp. 16–19). The advantages that can be secured include the following:

- *Cost savings* can be achieved through economies of scale and economies of scope. Examples include the savings possible by making use of the discounts available when purchasing large volumes or through the shared use of resources, such as warehouses or offices. Moreover, costs can be saved through cooperative planning processes, for example for planning procurement or deliveries (Martín Díaz 2006).
- *Time savings* can be achieved, for example during development work, by combining resources. This can accelerate time-to-market. Development partnerships can also reduce risk. For instance, by sharing the costs of development work, the risks of failure are shared and therefore minimized for the participants.
- Cooperation can also increase the *value of the product or service*. For example, airlines, car rental companies, and hotels form alliances to offer additional services, such as the synchronized provision and return of rental vehicles and the granting of bonus points. Open-source software is also developed through collaborative activities. The more programmers are involved, then the better the quality of the software, at least as a general rule. In his famous article, "The Cathedral and the Bazaar," Eric S. Raymond states "given enough eyeballs all bugs are shallow" (Raymond 1999).
- Cooperation and acquisitions can grant *access to new markets*. This can involve new geographies, or an extension to the existing product portfolio.

The potential advantages are ultimately reflected either in savings or (directly or indirectly) in an increase in revenues. Dividing up the value added between the partners can be a significant challenge. Many cooperative projects stumble at this particular hurdle—which is only trivial at first glance—even before they really begin. We wish to illustrate this problem using an example of cooperative game theory.

A rich man and a beggar are walking along the road, and both of them discover some money, let us say 100 euros, simultaneously. The challenge for the two of them is to divide up the money in a way that is satisfactory for both. If they manage to find a consensus, each gets to keep his portion of the find. If they fail to agree, both will be left empty-handed.

The problem is easy to understand, but a mutually agreeable solution is difficult to find. Both the beggar and the rich man want to keep as much of the money as possible for themselves. This particular constellation can often be seen in similar form between business partners. The theoretical solution of the problem is complex in mathematical terms, and is based on a variety of utility functions for the two parties (Sieg 2005, p. 181 ff.). On the basis of an assumed linear utility function for the rich man and a logarithmic function for the beggar, it is possible to arrive at a split of approximately 23 euros for the beggar and 77 euros for the rich man. However, this is the theory, and in practice it is very difficult to get the two parties to agree on a split of this kind.

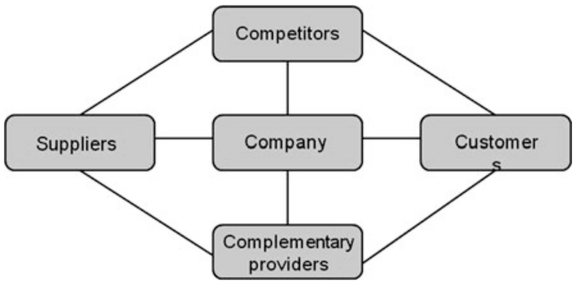


Fig. 3.1 Systematic definition of potential partners

We, therefore, wish to attempt to simplify the matter: we can assume that the allocation of the benefits of cooperation must be Pareto optimal. In other words, the gains must be divided up in a way that ensures that at least one of the partners is better off than before, but none is worse off. If this is not the case, then at least one of the partners will generally not wish to cooperate.

There are pragmatic alternatives, including the following simple approaches (Buxmann et al. 2007):

- The profits of cooperation are divided among *n* participants, so that each participant receives the *n*th share of this additional profit.
- The profits of cooperation are divided up in accordance with the pattern of profits prior to the partnership.

In the first instance, the smaller partners gain disproportionately. With the second model, the stronger participants tend to be at an advantage. Even if one or both of these possible models lead to Pareto optimality, this does not mean that the participants would actually come to an agreement. There are a wide variety of possible distribution mechanisms, with an equally large variety of advantages and disadvantages for the individual participants. Whatever happens, each participant will attempt to secure the largest possible slice of the available cake for themselves.

Moreover, any partnership entails a sizeable investment on the part of the participants. This begins with the costs of seeking the right business partner. Then there is the expense, often considerable, of contract negotiations. Negotiations will focus, for example, on the partners' contributions in terms of investments and their share of any profits. Moreover, there will be significant investment in the establishment of a shared infrastructure and in developing skills at the participating organizations (Hirnle and Hess 2006).

In the next section, we wish to concentrate on forms of cooperation within the software industry.

3.1.1.2 Types of Cooperation and Partners Within the Software Industry

First, let us consider who the potential partners for software companies are. In this context, we consider the Brandenburger and Nalebuff model, in Fig. 3.1 (Brandenburger and Nalebuff 1996, p. 17).

In other words, we assume that a company, in the automotive industry, the software industry, or in a different sector, has the four potential partners given above: customers, complementary providers, suppliers, and competitors.

Moreover, partnerships can be better understood by employing the classification created by Ralf Meyer, which differentiates between (Meyer 2008):

- Development partnerships,
- Reseller partnerships,
- Shared revenue partnerships,
- OEM partnerships,
- Referral partnerships, and
- Standardization partnerships.

When defining collaborative relationships, the following questions need to be addressed (Meyer 2008):

- Which partner is to deliver the product to the end-customer (who ships)?
- Which partner determines pricing and discounts (pricing)?
- How is the product to be branded (branding)?
- Which partner has the intellectual property rights to the product (IP)?
- Which partner will post sales revenue for the product (revenue booking)?
- Which partner is responsible for customer contact (customer control)?
- Which partner is responsible for go-to-market activities?
- Does the recipient carry out quality checks (quality assurance)?
- Which partner is responsible for support?

Development Partnerships

A development partnership is a collaborative relationship with the aim of creating new software and service solutions. Software providers can cooperate with any of the partners depicted in Fig. 3.1. The development tasks are shared among the partners, taking on the role of inventors.

A frequent example is the joint development of a product and systems on the part of standard software vendors and their customers. Generally, the aim is to develop a solution that supports industry-specific needs that are not modeled within the standard product. What are the advantages of this type of cooperation? The customer gains access to a software solution tailored to its particular needs. The software vendor gains insight into a particular industry. An example is the development of a solution for scheduling agreements with suppliers, created within the scope of a partnership between SAP and Bosch (Buxmann et al. 2004).

Frequently, as the following example illustrates, joint software development leads to the establishment of a joint venture.

iBS Banking Solution

Within the scope of a dedicated project, CSC Ploenzke created an in-house solution for the mortgages unit of DePfa Bank (since renamed Aareal Bank). The solution adds important components to the standard SAP solution for banks. Its functionality encompasses all lending and deposit-taking, integrated derivatives, and money market and foreign exchange trading. The solution also forms the basis for end-to-end banking management, including risk management, on the basis of the SAP system. Once the project was completed, the participants came to the conclusion that the software solution could be marketed to other companies. The two parties founded a joint venture, with CSC Ploenzke holding a 51 % stake and DePfa the remaining 49 %. There were two reasons for this move. First, there was an issue with the fact that potential customers were DePfa's direct competitors. Second, DePfa did not have sufficient consulting resources to implement the software at other organizations.

Sources www.ibs-banking.com; Sapinfo.net/SAP-BranchenmagazinBanken and Versicherungen, no. 3 March 2001, pp. 20–21.

Suppliers are an additional group of potential partners for joint development activities. In the software industry, these particularly comprise other software companies who can take responsibility for selected development tasks, or freelance contractors who can be incorporated into project teams. Major standard software vendors, for example, frequently work with a large number of software subcontractors. In particular, the trend toward service-oriented architectures (see Sect. 4.7.2) may facilitate the establishment of further collaborative arrangements at the interface between software vendors and subcontractors. This opens up new opportunities for niche players to offer software as a service (SaaS).

When a standard software provider finds that the costs of developing a particular service exceed the expected value added, the obvious solution is to outsource this development work to a software supplier. This limits the software provider's development risk and the supplier gains an opportunity to integrate its services into the major player's solution. To date, in most actual cases of these types of partnership, the provider and the supplier deal with the customer separately—and send their own invoices. In the future, closer collaboration will be conceivable and worthwhile. The partnership could be set up so that the suppliers will have a share in the revenues. A major challenge here, however, is to find the right formula to divide up the spoils. In addition, both partners must invest in the partnership. Typically, the supplier will have to make the larger investment, especially in the form of training costs. Often, suppliers and development partners must attend—often costly—training sessions organized by the software provider, in order to gain entry to particular partner programs. However, this is an understandable approach from the provider's perspective: For one thing, it ensures that suppliers are familiar with the relevant underlying technologies, and for another, training can generate considerable revenues.

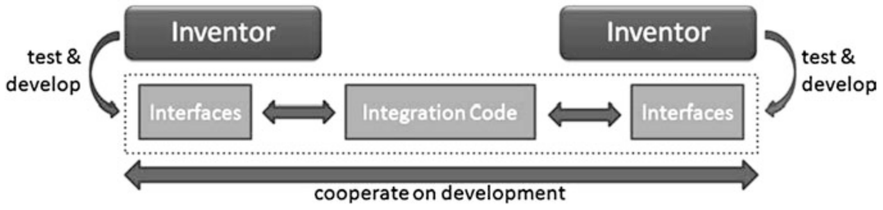


Fig. 3.2 Outline of joint development of integration components (based on Meyer 2008, p. 110)

This type of partnership would also be possible with *competitors*. This relationship is sometimes described as "co-opetition," because "you have to cooperate and compete at the same time" (Brandenburger and Nalebuff 1996, p. 4). The challenge here is to establish a win–win situation, where both partners profit from the arrangement, although they are and will remain competitors.

The potential advantages of co-opetition are more or less those already outlined in Sect. 3.1.1.1. One obvious motivator of cooperation is the opportunity to share resources. If at least one of the parties has idle development or production resources, while its competitors are operating at close to capacity, it makes sense to share those resources. The best-known examples of this form of cooperation are to be found in the automotive industry. For instance, Daimler and Volkswagen have long collaborated on the development of engines and commercial vehicles, while Porsche and Toyota are working closely together on the development of hybrid drive technology.

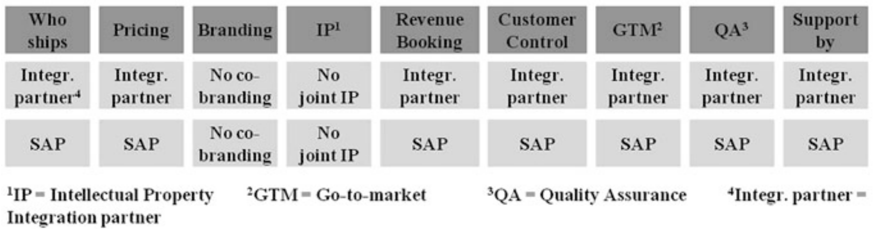
DUET is an example of cooperation between two major software providers that are competitors in some areas: Microsoft and SAP. DUET offers users an interface between Microsoft's Office applications and SAP's ERP systems. Microsoft and SAP offer competing enterprise software solutions for small and midsize companies, and both hope that the joint solution will deliver a win–win situation.

Payment for and allocation of intellectual property (IP) rights among the parties varies considerably from case to case. Figure 3.2 illustrates a type of development partnership that is often found in the SAP ecosystem. This concerns the joint development of integration components.

Figure 3.3 depicts another type of development partnership, exemplified by the integration of partner products into a SAP solution.

Reseller Partnerships

This model is characterized by a purchaser–provider relationship. In this case, the purchaser incorporates the provider's solutions in its product portfolio and resells them to its own customers. In other words, the provider produces the software and sells licenses to purchasers, taking on the dual role of inventor and—from the purchaser's perspective—IP lessor (see Sect. 1.4). The purchaser also occupies the role of IP lessor, as it resells the licenses it has bought. Figure 3.4 provides an outline of a reseller partnership.



Who ships	Pricing	Branding	IP[1]	Revenue Booking	Customer Control	GTM[2]	QA[3]	Support by
Integr. partner[4]	Integr. partner	No co-branding	No joint IP	Integr. partner	Integr. partner	Integr. partner	Integr. partner	Integr. partner
SAP	SAP	No co-branding	No joint IP	SAP	SAP	SAP	SAP	SAP

[1]IP = Intellectual Property [2]GTM = Go-to-market [3]QA = Quality Assurance [4]Integr. partner = Integration partner

Fig. 3.3 Development partnership–parameters (exemplified by an SAP integration project) (based on Meyer 2008, p. 76)

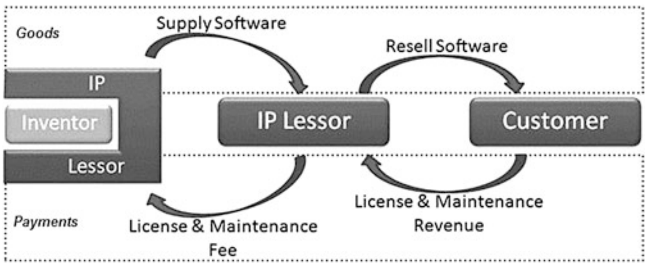
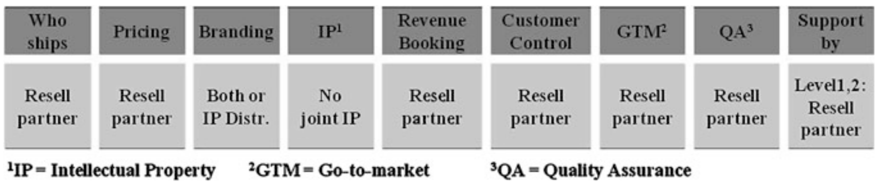


Fig. 3.4 Outline of reseller partnership (based on Meyer 2008, p. 78)



Who ships	Pricing	Branding	IP[1]	Revenue Booking	Customer Control	GTM[2]	QA[3]	Support by
Resell partner	Resell partner	Both or IP Distr.	No joint IP	Resell partner	Resell partner	Resell partner	Resell partner	Level1,2: Resell partner

[1]IP = Intellectual Property [2]GTM = Go-to-market [3]QA = Quality Assurance

Fig. 3.5 Reselling–parameters (based on Meyer 2008, p. 76)

Reseller partnerships are commonly found in the case of software providers and their sales partners. But they also occur between suppliers and providers.

For suppliers, the potential advantage of this arrangement is that they obtain access to the provider's customers, without having to establish their own sales organization. While coordinating the partnership involves some additional costs, these are likely to be offset by the increased revenues from the sales channel.

Reseller partnerships are usually organized as shown in Fig. 3.5.

Shared Revenue Partnerships

This type of partnership involves a software provider selling its products through a broker. The latter may also be a platform. Above all, the broker allows the provider to benefit from its strong market position by means of a joint market presence.

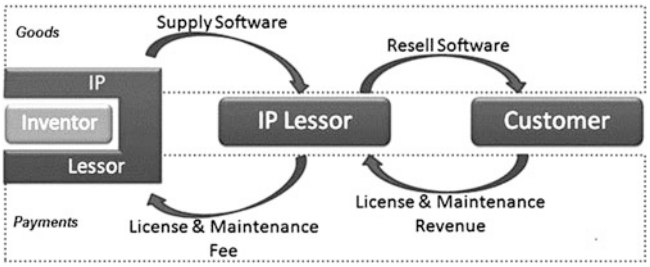


Fig. 3.6 Outline of shared revenue partnership (based on Meyer 2008, p. 86)

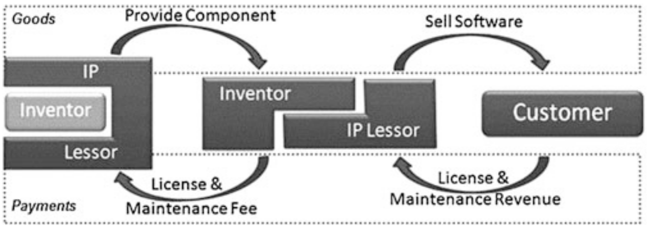


Fig. 3.7 Outline of OEM relationship (based on Meyer 2008, p. 97)

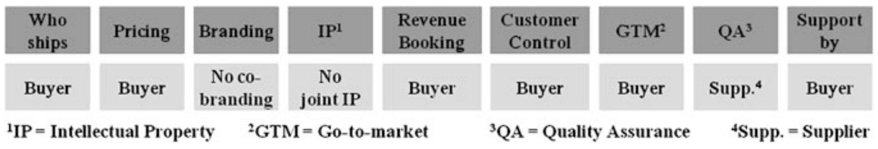
In contrast to resellers, brokers do not assume any proprietary rights to the software products they sell. Instead, they merely recommend the third-party solution to their customers, fulfilling the role of an IP broker. This model is illustrated in Fig. 3.6.

Software distribution via mobile application portals is a current example of this type of cooperation: Software providers—either companies or individual developers—can use these platforms to provide customers with software solutions and/or apps for their mobile devices. The best-known examples are Apple's App Store and Google's Android Market.

OEM Relationship

The OEM model is essentially the traditional buyer–supplier relationship, as found in the automotive industry, for example. In other words, a company integrates components or systems developed by suppliers into its own product. In the software industry, OEM relationships involve both suppliers and buyers acting as inventors and IP lessors (Fig. 3.7). In rare cases, buyers can even assemble a solution entirely from supplied components. In this scenario, the buyer's role as an inventor is limited to the activities required to integrate the components.

This model enables suppliers to benefit from higher sales revenues and greater market penetration. Often, this type of relationship consists of smaller companies providing larger ones with components. But it can also look completely different. For example, SAP frequently acts as an OEM partner by providing its NetWeaver platform to other companies. On this basis, the latter can develop complementary solutions and sell them as a package to their own customers.



Who ships	Pricing	Branding	IP[1]	Revenue Booking	Customer Control	GTM[2]	QA[3]	Support by
Buyer	Buyer	No co-branding	No joint IP	Buyer	Buyer	Buyer	Supp.[4]	Buyer

[1]IP = Intellectual Property [2]GTM = Go-to-market [3]QA = Quality Assurance [4]Supp. = Supplier

Fig. 3.8 OEM relationship–parameters (based on Meyer 2008, p. 76)

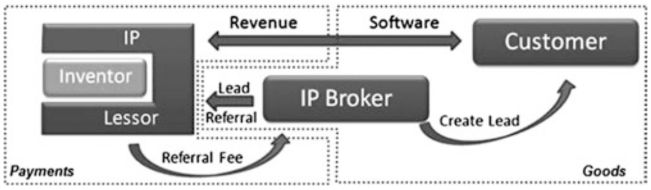
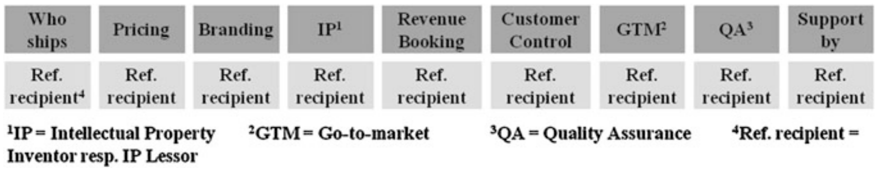


Fig. 3.9 Outline of referral partnership (based on Meyer 2008, p. 93)



Who ships	Pricing	Branding	IP[1]	Revenue Booking	Customer Control	GTM[2]	QA[3]	Support by
Ref. recipient[4]	Ref. recipient	Ref. recipient	Ref. recipient	Ref. recipient	Ref. recipient	Ref. recipient	Ref. recipient	Ref. recipient

[1]IP = Intellectual Property [2]GTM = Go-to-market [3]QA = Quality Assurance [4]Ref. recipient = Inventor resp. IP Lessor

Fig. 3.10 Outline of referral partnership (based on Meyer 2008, p. 76)

From the buyer's perspective, the main advantage of the OEM model is that purchasing components and solutions cuts development time, costs, and risks. Suppliers can deliver cost savings because they generally sell their products to multiple buyers, enabling them to distribute development costs across several parties.

A typical OEM relationship is organized as shown in Fig. 3.8.

Referral Partnerships

Like shared revenue models, referral partnerships involve an IP broker (Fig. 3.9). In the following, we describe the latter as the "referral provider." This partner sells information about potential customers (leads) to the referral recipient (IP lessor). In return for this information, the IP broker receives a combination of a fixed fee and a share of the resulting revenues. As the referral provider mostly has no direct influence over these customers, but merely passes on their contact details, the share is generally a lot smaller than in the shared revenue model.

Figure 3.10 shows how this partnership is organized.

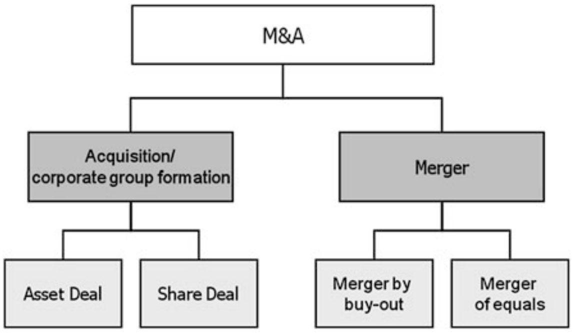


Fig. 3.11 Types of mergers and acquisitions (Wirtz 2003, p. 13)

Standardization Partnerships

Standardization partnerships are somewhat different: As a general rule, no money changes hands between the parties. However, they are an important element in many software and IT enterprises' strategies. These companies frequently form strategic alliances. Mostly, their aim is to establish standards—or to prevent the widespread adoption of standards that would benefit potential competitors.

The working groups of standardization organizations, such as the World Wide Web Consortium, frequently facilitate this type of cooperation. Participating companies aim to give input into the technical specifications of the new standard. But they also know that they need to be inside the tent to ensure that their own or preferred standard is chosen over others, possibly favored by competitors. We discussed the importance of setting standards in some detail in Sect. 2.2.

3.1.2 Mergers and Acquisitions in the Software Industry

In this section, we will focus on mergers and acquisitions (M&A) in the software industry. These play a particular role in the software industry, because network effects make the size of a provider and its network a crucial competitive advantage. First, we will consider the various forms of M&A (Sect. 3.1.2.1). This will feed into an investigation of the various motives for acquisitions (Sect. 3.1.2.2), before we turn to the consolidation trend in the software industry (Sect. 3.1.2.3). Finally, we will analyze the success of M&A in the software industry (Sect. 3.1.2.4).

3.1.2.1 Forms of M&A

There is a multitude of definitions surrounding M&A. We do not intend to go into them here. However, a common definition is that acquisitions involve at least one company relinquishing its financial—and possibly also legal—independence (Wirtz 2003, p. 15).

A distinction can be made between M&As, as shown in Fig. 3.11. Mergers entail the fusion of two independent companies into one legal entity. In other

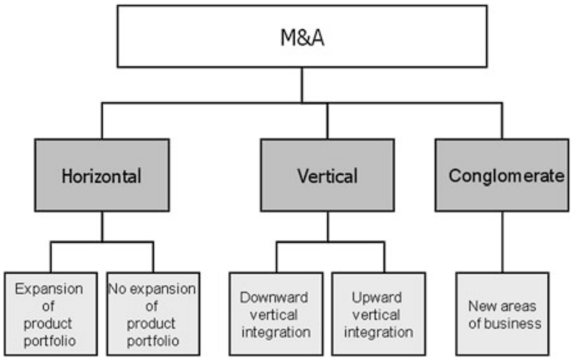

M&A
Horizontal
Vertical
Conglomerate
Expansion of product portfolio
No expansion of product portfolio
Downward vertical integration
Upward vertical integration
New areas of business

Fig. 3.12 Forms of M&A (Wirtz 2003, p. 19)

words, both parties surrender their legal independence, although they can opt to form a new legal entity or be incorporated into an existing one. In acquisitions, by contrast, one company is integrated into a corporate group—this does not necessarily involve a legal fusion.

Another distinction can be drawn between horizontal, vertical, and conglomerate (also known as diagonal) M&A (Wirtz 2003, p. 18 ff.).

Horizontal M&A involve companies in the same industry at the same stage of the value chain. They are generally intended to increase competitiveness and realize synergies in the form of economies of scale and/or economies of scope. Examples of horizontal mergers in the software industry include the acquisition of PeopleSoft by Oracle (see Sect. 3.1.2.3) or Symantec's purchase of storage and security solution provider Veritas, for some $13.5 billion (Parbel 2005).

Vertical M&A combine companies on different levels of the value chain. In other words, an enterprise joins forces with another that is directly upstream or downstream. This is also known as upward or downward vertical integration. The aims behind vertical M&A are cutting transaction costs, improving planning across the value chain, and better access to procurement (in the case of downward vertical integration) or purchasers (in the case of upward vertical integration). The expansion of Software AG in Latin America is an example of this type of deal. In 2005, the German provider acquired APS Venezuela and five affiliated companies in Panama, Costa Rica, and Puerto Rico. Previous to this, APS Venezuela had been Software AG's sales partner and an established distributor of transaction systems for major accounts in the financial, manufacturing, oil and mining industries, and the public sector. The move was intended to strengthen Software AG's presence on the Latin American market.

Diagonal or *conglomerate* M&A bring together companies from different industries or segments, enabling the penetration of new markets. These generally result from a diversification or expansion strategy. For example, this is the strategy followed by Infor. Its sales revenue of approximately $2.1 billion and headcount of more than 8,000 make it one of the world's largest software companies. Infor's strategy aims for rapid growth by buying up a number of smaller software providers from a variety of fields. In contrast to many other acquisitions in the industry, those made by Infor are not primarily driven by migration strategy: the company does not intend to integrate the acquired applications into a single end-to-end solution. In addition to licensing, Infor quickly generates income from the target companies' on-going service contracts. Moreover, it can take advantage of cost-cutting opportunities such as tighter cost management or by consolidating corporate or administrative functions.

Figure 3.12 illustrates the various forms of M&A.

In the following section, we will look at the motivations that prompt management to acquire another company or sell their own.

3.1.2.2 Motivations for M&A

As M&A can affect every aspect of the companies involved, any examination should theoretically address the viewpoints of all relevant stakeholders. This would include shareholders, management, employees, suppliers, customers, competitors, and even society itself, as M&As frequently cause significant changes in the labor market.

In the following section, however, we will concentrate on a brief outline of the reasons from management's perspective. These can be grouped into strategic, financial, and personal motivations (Wirtz 2003, pp. 57–76).

Strategic motivations are generally about realizing synergies and can be divided into

- Market motivations,
- Performance motivations, and
- Risk motivations.

Market motivations relate to both the procurement and sales sides. For example, the deal may increase negotiating power over mutual suppliers, as larger quantities are at stake. In addition to procurement, boosting sales is another key motivation for M&A. By combining their sales activities, the companies involved can realize synergies and gain competitive edge. A stronger position on the sales market can bestow greater influence over prices, and can even help to squeeze competitors out. Finally, an acquisition can also offer growth potential, for example in the form of access to new regional markets.

Synergies may also be created by pooling resources and skills. *Performance motivations* are the expectation of synergies in corporate activities such as research and development, procurement, production, marketing. For one thing, the companies' technologies and expertise can be combined to produce new or better quality products and services. In the context of software, the development of integrated systems is conceivable. In addition, the deal could lead to a better utilization of existing development resources.

Risk motivations are most commonly found in relation to M&A activities driven by a diversification strategy. For example, a company may see risks in its dependence on a particular product or the development of a particular industry.

Table 3.1 Comparison of M&A activities by industry, 2009 (USA) (Mergerstat Free Reports 2009)

Rank	Industry	Deals	Value in millions of $
1	Drugs, medical supplies, and equipment	319	219,089.7
2	Computer software, supplies, and services	1338	50,911.6
3	Brokerage, investment and management, and consulting	521	47,170.5
4	Energy services	83	42,586.3
5	Transportation	81	29,626.0
6	Banking and finance	272	28,632.9
7	Food processing	83	22,665.7
8	Miscellaneous services	816	19,636.7
9	Broadcasting	73	19,069.0
10	Chemicals, paints, and coatings	106	14,060.0
Total Top 10		3692	493,448.4

Expanding the product portfolio or tapping into new industries by means of an acquisition is seen as a way of reducing these risks.

It should be noted that the strategic motivations listed above also apply to cooperation in general, and should be regarded as supplementing the list of advantages of cooperation in Sect. 3.1.1.1.

In addition to the above-mentioned market, performance and risk motivations, there may also be *financial motives* for a merger or acquisition. The most powerful motivation is generally raising profitability by generating profits or by taking advantage of tax losses carried forward. These options will be based on considerations relating to developments on capital markets, balance-sheet optimization, and taxation.

Moreover, management may also have *personal motivations*. A number of explanations have been put forward, including managers overestimating their own abilities and empire-building, as the real reasons behind some acquisitions (Wirtz 2003, pp. 57–76).

3.1.2.3 Consolidation Tendencies in the Software Industry

We have already noted a number of times that, due to network effects, software markets are governed by the winner-takes-all principle. M&A help fuel this tendency toward the establishment of monopolies. Their significance in the software industry is highlighted in the following table, which compares M&A activities across industries in 2009. Out of 49 industries studied, the tables show the top 10 on the US (Table 3.1) and European markets (Table 3.2). In the USA, the software industry occupies second place, in terms of transaction volume. By way of comparison: in 2006, the software industry was ranked sixth (Buxmann et al.

Table 3.2 Comparison of M&A activities by industry, 2009 (Europe) (Mergerstat Free Reports 2009)

Rank	Industry	Deals	Value in millions of $
1	Drugs, medical supplies and equipment	71	25,690.5
2	Brokerage, investment and mgmt., and consulting	86	22,535.7
3	Food processing	20	19,939.3
4	Insurance	35	12,560.7
5	Broadcasting	18	8,955.2
6	Computer software, supplies and services	204	8,755.1
7	Beverages	22	7,059.6
8	Mining and minerals	16	6,044.5
9	Electrical equipment	19	6,040.5
10	Miscellaneous services	144	4,587.6
Total Top 10		635	122,168.7

2008a). If we look at the number of deals, though, the software industry topped the rankings in both 2006 and 2009. In Europe, the software industry is sixth in terms of transaction volumes. But with respect to the number of deals, the industry comes out on top here as well.

A well-known example of the consolidation trend is the ERP software segment, most notably the takeover strategies pursued by Oracle. In recent years, Oracle has attracted attention for its many M&A activities.

On June 6, 2003, just 4 days after PeopleSoft disclosed its intention to acquire competitor J. D. Edwards, Oracle announced its own plan to buy up PeopleSoft for $5.1 billion. The acquisition of J. D. Edwards would have made PeopleSoft the second largest provider of enterprise software after SAP. Oracle's announcement triggered a 19-month long takeover battle. Finally, Oracle emerged victorious, securing its rival for some $10.3 billion. This made it the industry's largest merger to date. Oracle was restored to its runner-up position on the enterprise software market, and was able to edge closer to market leader SAP.

Oracle promised not only to continue supporting PeopleSoft and J.D. Edwards' software until 2013, but also to develop it further. At the same time, it released a new, integrated software solution called "Fusion," combining all product lines. This initiative must have required a significant effort on Oracle's part; Fusion's R&D team is already one of the world's largest.

Oracle then added one of the leading providers of CRM software to its holdings, acquiring Siebel for around $5.85 billion. This intensified the wave of consolidations on the ERP software market.

Recently, Oracle purchased Sun Microsystems, giving it a foothold on the hardware market.

A hotly debated theory posits that M&A activity upsets the market's equilibrium: it essentially forces other organizations to make their own acquisitions, in order to preserve their long-term independence on the market and avoid becoming a takeover target themselves (Hutzschenreuter and Stratigakis 2003).

In the software industry in particular, network effects mean that size is in itself an advantage for providers, giving them an edge over the competition. This was the reason given by Oracle CEO Lawrence Ellison, who stated that the acquisition of Siebel would "strengthen our number one position in applications in North America and move us closer to the number one position in applications globally." Moreover, in this case, Oracle also had the opportunity to offer its customers an end-to-end integrated solution.

A similar consolidation trend can be seen on the market for office software. A few years ago, there were a number of viable options, such as Lotus 1-2-3 for spreadsheets or Word Perfect for word processing. Now, Microsoft has a virtual monopoly over this market. The open source community is the only remaining serious competitor (see Chap. 7).

Even though monopolistic structures are often regarded as disadvantageous for the buyers, their impact on network markets is a matter of debate. On the one hand, of course, customers are far more dependent on the provider. Theoretically, this would enable the provider to raise prices, which—as shown above—has not happened in the case of Microsoft. An additional drawback for users is that monopolies tend to produce fewer innovations. On the other hand, monopolistic structures have the advantage of avoiding incompatibility issues. This is the line taken by Stanley Liebowitz and Stephen Margolis in various articles (e.g., Liebowitz and Margolis 1994, 2001).

Empirical studies show that the wave of consolidations in the software industry is primarily being driven by leading players (Friedewald et al. 2001). One response open to smaller software providers is to cooperate with other software companies; another is to concentrate on specialist or niche solutions.

3.1.2.4 Factors Determining the Success of M&A Activities in the Software Industry

On software markets, M&As give companies the opportunity to gain strategic competitive advantage. As previously explained at some length, this is reinforced by the fact that software providers operate on network effect markets. But do acquisitions really increase the value of the purchaser's company? This question was addressed by Izci and Schiereck in an empirical study (Izci and Schiereck 2010).

The dataset comprised 81 international acquisitions in the enterprise software industry between 2000 and 2007. All of these transactions were worth at least \$50 million and the purchaser acquired at least 20 % of the target company. Both the purchaser and the target were listed companies, whose stock prices could be tracked between 230 days before and 30 days after the announcement of the deal.

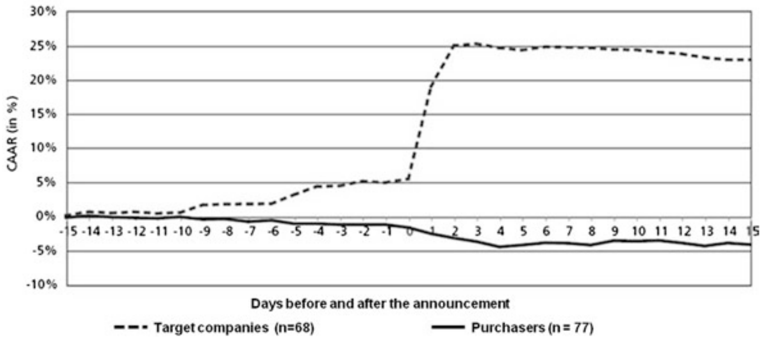


Fig. 3.13 Cumulative average abnormal return (CAAR) for purchaser and target companies (Izci and Schiereck 2010)

The investigation was based on the analysis of abnormal returns. These are the difference between actual return and the "normal" return that would have been expected had there been no M&A announcement. The abnormal returns for the purchaser and target were calculated and cumulated over a period comprising up to 15 days before and after the M&A announcement.

The results showed that target companies obtain a positive cumulative abnormal return (CAR), while the purchasers' value trends to dip slightly. The CAR experienced a statistically significant rise or fall, particularly when the announcement was made and a few days following it, and the trend continued throughout the observation period thereafter (see Fig. 3.13).

In a second step, the researchers looked at the factors affecting the development of the stock price, and therefore the success of the purchaser. On the basis of a multivariate regression, they found no evidence that acquiring a company from the same industry or segment or in the international environment had the expected positive effect (Izci and Schiereck 2010).

While the analysis did show a small positive correlation, it was not statistically significant. Similarly, the percentage of stock acquired and transaction volume had no meaningful effect on the purchaser's value. The only factor that showed a slightly positive impact was cash rather than stock purchase: Buying with cash had a positive effect on the stock market valuation.

By contrast, there were significant negative impacts in relation to the relative size and stock market performance of the target company. The stock market reacts unfavorably to major acquisitions, speculating that the integration costs incurred will outweigh the potential synergies (Loefert 2007, p. 163; Izci and Schiereck 2010).

The results of the stock market study show that M&A transactions in the enterprise software industry add value (at least initially) for target companies only, while purchaser companies are at a disadvantage, in terms of shareholder value.

What is surprising is that overall high risk, in what is an innovative and dynamic segment, does not appear to be responsible for the negative valuation of the purchaser. Cross-border or diversifying transactions are not regarded as particularly negative. Instead, it is cultural challenges that are likely to be punished by the stock market. To earn a positive reaction from the markets, enterprise software providers will have to convincingly demonstrate that the integration of their major competitors will result in strong growth without risking synergies. The alternative—foregoing acquisitions altogether—is not a realistic option in a rapidly consolidating network market like the software industry.

3.2 Sales Strategies

Sales strategies involve decision-making around the provision of goods and/or services to companies in the downstream value chain. This comprises activities that directly address prospects, and sales logistics activities (Homburg and Krohmer 2006, pp. 864–866).

This section discusses channel management activities; sales logistics is about ensuring that physical products reach the end-customer, and is therefore not relevant to software products.

In channel management, the following issues are of central importance:

- How the sales system is structured?
- How relationships with sales partners and key accounts are managed?
- The use of Key Performance Indicators (KPIs) systems in sales management and
- How sales activities are organized?

3.2.1 Structuring of Sales Systems: Organization and Sales Channels in the Software Industry

Structuring a sales system involves taking decisions about the sales organization and sales channels.

Let us begin by looking at the sales organization. This can generally be structured according to:

- Region,
- Industry,
- Product as well as
- New and existing customers.

Structuring sales by region generally means by continent, country or federal state. The advantage of this approach is that it ensures a measure of geographical proximity to the customer.

But sales can also be organized by industry. The advantage here is that the sales professionals have the relevant industry-specific skills and 'speak the customer's language.' One drawback is that they typically spend more time traveling.

Structuring by product is the prime strategy of software companies that have a broad range of products and whose sales activities rely on product-specific skills.

The sale of complex SCM or CRM systems is an example of this. Against this background, many software companies structure their sales and consulting business not only by region, but also by product. In the SCM and CRM field, for example, many software companies have experts who address highly specialized issues in worldwide projects, such as optimization algorithms for the implementation and use of SCM solutions.

It makes sense to structure the sales organization in terms of new and existing customers because these two target groups require different types of sales employees, who can be labeled ‘hunters’ and ‘farmers.’ As the name suggests, the hunter type is best suited to winning new customers and selling them a new product or solution. The farmer type, on the other hand, feels more comfortable working with customers with whom he has built up a relationship over a long period of time. This type should, therefore, be deployed in sales to existing customers.

The above criteria can be combined, of course. It is common practice, for instance, to create roles defined by product and region. This leads to one department being responsible for specific continents and products. The advantage is that the unique characteristics of specific regions and products are fully taken into account, but the disadvantage is that it is not always clearly defined who is responsible for what.

In the following, we will look at how to structure *sales channels*. The most basic decision to be made is whether to choose direct or indirect sales. Indirect sales are when sales activities are performed by third parties, such as VARs and system integrators. With direct sales, these tasks are carried out in-house (Homburg and Krohmer 2006, pp. 873–877).

When choosing between direct and indirect sales, businesses should take into account both efficiency and effectiveness considerations. In terms of efficiency, they should evaluate the two options in light of the associated transaction costs (see Sect. 2.4), as using sales partners can lead to savings in this area. Those savings are comparable with a trade margin in license sales. Effectiveness considerations can relate to customer service quality, for example regarding geographical proximity or specialization, or customer allegiance. For instance, VARs and systems integrators might concentrate on specific industries and develop deep skills. In addition, businesses in fast-growing sections of the software industry tend to work with sales partners, because they could not achieve rapid growth without them.

We will now look at some factors that can influence the advantages of direct versus indirect sales. There is an assumption in the marketing literature that a high specificity and product complexity would favor direct sales (Homburg and Krohmer 2006, p. 874).

But does this necessarily apply to the software industry? With regard to specificity, the assertion is correct. A provider of custom software, i.e., specific solutions, is not likely to sell those solutions via indirect channels. So, indirect sales are only an option for standard software providers. However, these products are often extremely complex. As we saw in Chap. 1, this applies especially to ERP systems, considering the huge amount of customization work required during implementation projects. Nevertheless, standard software providers frequently opt for indirect sales. But this route calls for sales partners with a correspondingly high level of (often industry-specific) expertise. So, as far as the software industry is concerned, high specificity signals a tendency to use direct sales. A further advantage of direct sales, of course, is that it enables companies to build customer allegiance and intimacy.

The choice of direct versus indirect sales will also depend on how many potential customers there are. The advantages of indirect sales tend to increase with the number of customers. Again, this can be explained in terms of the transaction costs. As we saw in Sect. 2.4.4 in the discussion of intermediaries, savings potential rises with the number of market participants.

Now, one might argue that the availability of the Internet and the characteristics of software as a good would favor direct sales. Why would the provider not simply sell the software direct to customers over the Web? This is no problem when the product is relatively easy to understand, such as antivirus software. But it is quite a different thing when it comes to implementing a complex solution, one with comprehensive supply chain management functionality, for example. In this case, the customer is certain to require a good deal of advice and support.

We underlined the international nature of the software industry at the beginning of the book. This affects the structure of sales in various ways. A study by Lünendonk showed that companies generally conduct indirect sales in other countries via a subsidiary. The second most common option is the use of collaboration partners, such as IT consulting companies, VARs and system integrators (Lünendonk 2007, p. 57).

However, as a rule, providers do not have to opt for exclusively direct or only indirect sales. They can often combine the two. SAP's sales strategy is a case in point. SAP divides up the market and as a result, differentiates its sales strategy in terms of

- Global Enterprises with at least 2,500 employees,
- Local Enterprises with 1,000–2,499 employees,
- Medium Enterprises with 100–999 employees, and
- Small Enterprises with 1–99 employees.

In a presentation to investors in January 2007, SAP offered the following market breakdown:

- 20,000 companies in the Global Enterprise segment,
- 1.3 million companies in the Local and Medium Enterprise segment,
- 55.4 million companies in the Small Enterprise segment.

Table 3.3 Availability of IT skills in midsize companies

Resources	Lower mid-market	Upper mid-market
IT department	None	In-house
ERP software skills	None	Yes
IT budget	None	Yes
Software decision makers	Senior management	IT department
Process complexity	Low	High
Number of users	Small	Large

SAP sells direct to global enterprises, i.e., it does not involve any partners in the sales process. The installed base in this segment ensures repeat sales and a steady stream of license income through upgrades, new releases, and maintenance.

As SAP already has a very large market share among the top 500 firms, its future growth will depend mainly on establishing a strong position among small and midsize enterprises. In line with this objective, the company launched the SaaS solution, SAP Business ByDesign. A major challenge is that the market segments targeted differ markedly in terms of structure. Moreover, the various industries represented by midsize companies make quite different demands on business software. As a result, SAP's Sales and Consulting units responsible for the lower mid-market are locally organized, whereas those for the upper mid-market are active throughout the whole country or even globally. These midsize enterprises can be very different with respect to the IT skills at their disposal, as the parameters in Table 3.3 describe.

SAP aims to acquire new customers in these segments by means of preconfigured industry-specific solutions with attractive entry-level pricing. In collaboration with its partners, the software giant is utilizing a new sales and implementation strategy: the try-run-adopt model, which allows companies to try out the software free of charge. In addition, the solutions are available as a SaaS offering (see Chap. 6). The first SAP partners began to market these industry solutions as packages in late 2006. With SAP's support, they offer customers a bundle comprising licenses, maintenance services, implementation, and the day-to-day management of SAP systems at a monthly rate that is currently well below € 200 per user. TV commercials have been used to promote this model.

When determining the shape of their sales system, providers must also decide on

- The length of the distribution channel,
- The width of the distribution channel, and
- The width of the sales system (Homburg and Krohmer 2006, pp. 877–884).

The length of the distribution channel indicates how many sales partners there are between the provider and the customer. In the software industry, multitier distribution channels make little sense. The advantages to be gained from them relate mainly to logistics, particularly warehouse storage, which are of little relevance to the software industry.

The width of the distribution channel expresses the number of sales partners a provider works with. In principle, it makes more sense to work with a small group of partners when the products sold are complex and of high value. As mentioned earlier, this is why software companies provide training to assure the quality of their sales partners—although another motivation is to generate sales in this sector themselves.

The width of the distribution system describes whether a product is distributed over just one channel or several. A single-channel system is one in which the product can only reach the customer by one channel. With a multi-channel system, the provider employs several channels to the customer. The music industry is one example of this approach: albums are sold via conventional brick-and-mortar retailers, online stores, or digitally via distributors such as Apple iTunes or Musicload.

Key goals of a multichannel system are broad market coverage and the ability to reach different customer segments. A core challenge is to create channels that complement and do not cannibalize each other. For instance, a publisher could offer subscribers of its daily newspaper various online functions that complement the print edition, such as search functionality or multimedia content like documentary films on featured topics. In the software industry, relatively self-explanatory products are sold both on the Internet and through intermediaries. However, cannibalization is less of a problem than in the music industry or in publishing, as software providers generally have no preference as to whether their sales are generated by direct or indirect channels.

When designing distribution systems, businesses need to clearly define the target groups and the tasks of the various distribution channels.

3.2.2 Organization of Relationships with Sales Partners and Key Accounts

We will now turn our attention to a software provider's relationship with its sales partners and key accounts. The term 'key account' denotes customers—usually enterprises—who are particularly important to the provider and who are therefore offered special deals or services.

Let us begin by looking at the provider-sales partner relationship. As in a supply chain, this relationship is based on the purchase by sales partners of the provider's goods and services—in our case, software licenses—in order to sell them on to their customers. The following section illustrates SAP‘s sales partnerships.

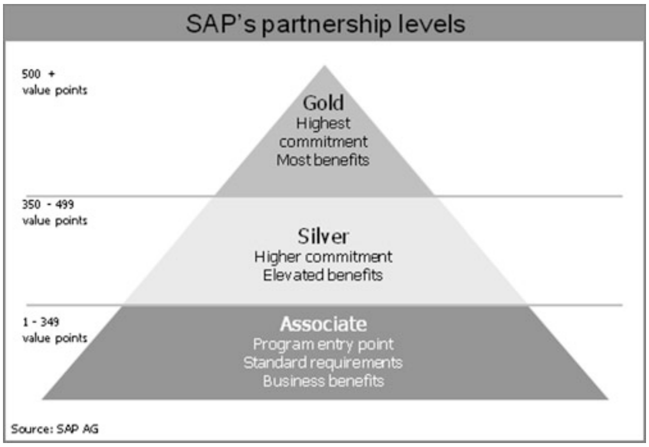


Fig. 3.14 SAP's partnership levels

Sales partnerships in the SAP space

Among SAP's partners, the mid-market is fiercely contested. To provide transparent information about the quality of its partners, SAP introduced a new method at the 2005 SAPPHIRE conference, whereby partners are granted a particular status (associate, silver or gold) via a scoring system known as Value Points (see Fig. 3.14). This system evaluates both the partner's skills and its portfolio. While sales success is the main criterion, customer satisfaction, training activities, the development of industry-specific solutions and add-ons also earn companies bonus points. According to the Value Points system, the score required for a particular status must be achieved over the past four quarters. Gold status is worth attaining for several reasons: partners with this status enjoy the biggest discounts on SAP licenses, which means they can achieve the highest margin when they resell them. An immaterial benefit is that the partner has close ties with SAP. Ultimately, the partner status reflects the development of the business relationship between SAP and the partner. A high status is bestowed in recognition of the partner's investment in the skills and resources needed to develop and implement SAP solutions.

SAP also provides partners with an extensive e-learning portfolio, and offers them sales and product training, which again earn them value points. Other concrete support includes quarterly assessments and reviews, strategy workshops, and joint marketing activities.

Key account management is crucial for software providers in the narrow and in the wider sense. That is why it is not unusual for executive board members or senior managers to be involved in this task. Managers often draft internal key account development plans that set out what revenues the company wishes to generate with which solutions. The discounts granted to key accounts, e.g., on software licenses, are normally considerably greater than regular discounts.

A primary goal of key account managers—regardless of the person's level in the hierarchy—is to be integrated in the customer's strategic planning. This can include jointly planning updates to a new release or the implementation of new, innovative technologies. Coordination can also be worthwhile on an operational level, e.g., regarding the customer's spending plan. To help key account managers with their tasks, some CRM systems enable them to create customer maps that include information about which customer employees are well-disposed to the provider and which are not. This information can be particularly vital when problems need to be communicated.

Organizing regular events for key accounts is another good way of improving communications and fostering customer allegiance. These can include invitations to sport and/or cultural events.

Before we examine the organization of sales processes, we would like to start by offering some thoughts on the management of sales activities. Performance measurement systems can be an important tool for this purpose.

3.2.3 Key Performance Indicators as a Sales Performance Management Tool in the Software Industry

3.2.3.1 Sales Performance Management

Sales performance management is the targeted management and coordination of a company's sales activities. We believe that performance management systems must do more than simply monitor performance by comparing target to actual figures. In a broad field of the literature, the main task of a sales performance management system is seen as the coordination of the management system. A coordination task always involves producing and utilizing decision-relevant information for the purpose of managing a business or part of a business—in our case, sales activities. A wide range of analysis tools is available to help supply pertinent information (Fig. 3.15).

In this section, we will restrict ourselves to showing how performance measurement systems are used, because this allows us to identify some special features of sales activities in the software industry. The use of other tools, by contrast, does not differ significantly between industries.

Use \ Analysis tool	ABC analysis	Portfolio analysis	Cost-performance analysis	Investment analysis	KPIs/performance measurement systems
Provision of information	X	X	X	X	X
Planning	X	X	X	X	X
Monitoring			X		X

Fig. 3.15 Selected analysis tools for sales performance management (Homburg and Krohmer 2006, p. 1214)

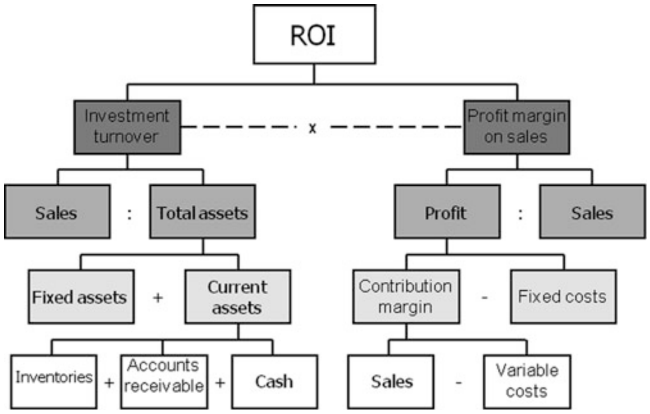


Fig. 3.16 The DuPont ROI model (Küpper 2008, p. 369)

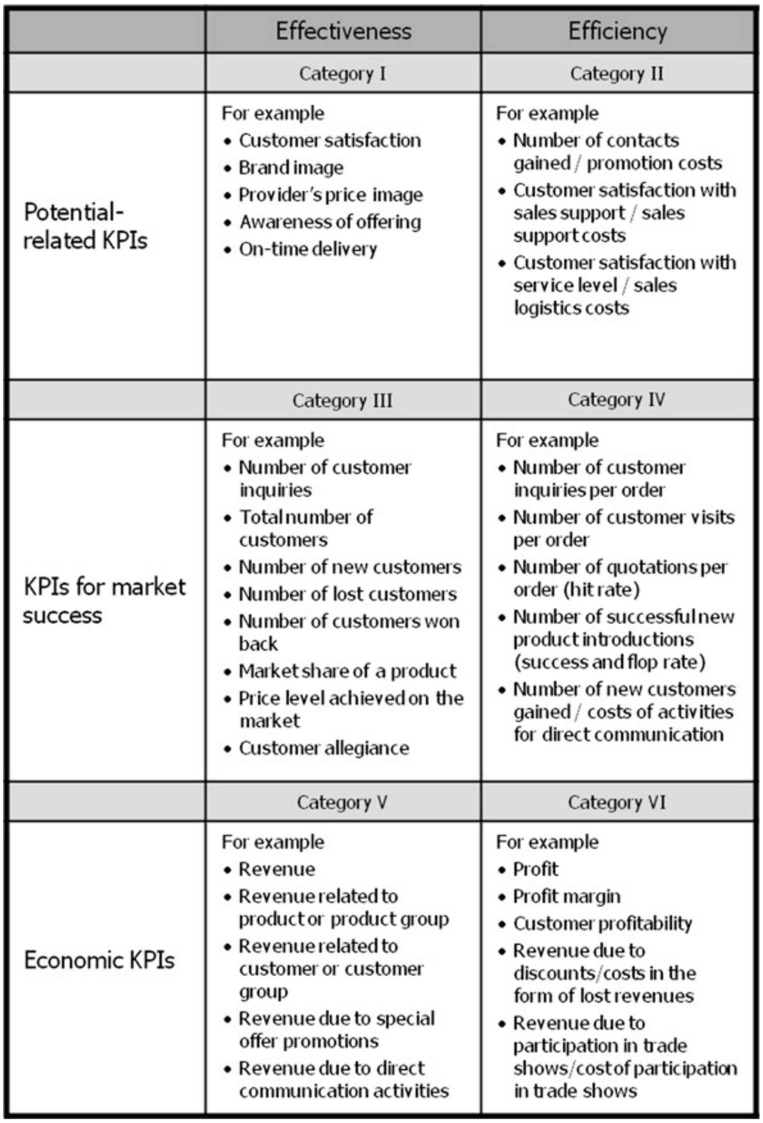
3.2.3.2 Use of Performance Measurement Systems in the Software Industry

Generally speaking, the purpose of KPIs is to evaluate something in quantitative terms. In the business world, performance indicators are used to provide management with decision-relevant information, usually for planning and monitoring. For example, KPIs can be used to specify targets for a given period, and at the end of that time to determine to what extent they have been met. If the actual figures differ from the targets, corrective action must be taken.

A variety of performance measurement systems has been developed, and they include many indicators that are related to each other. A classic example is the DuPont Return on Investment model shown in Fig. 3.16.

Modern management systems, such as the Balanced Scorecard, are also based on performance indicators as a tool for managing performance.

While the performance measurement systems discussed here are largely application independent, Homburg and Krohner developed a classification of KPIs



	Effectiveness	Efficiency
	Category I	Category II
Potential-related KPIs	For example • Customer satisfaction • Brand image • Provider's price image • Awareness of offering • On-time delivery	For example • Number of contacts gained / promotion costs • Customer satisfaction with sales support / sales support costs • Customer satisfaction with service level / sales logistics costs
	Category III	Category IV
KPIs for market success	For example • Number of customer inquiries • Total number of customers • Number of new customers • Number of lost customers • Number of customers won back • Market share of a product • Price level achieved on the market • Customer allegiance	For example • Number of customer inquiries per order • Number of customer visits per order • Number of quotations per order (hit rate) • Number of successful new product introductions (success and flop rate) • Number of new customers gained / costs of activities for direct communication
	Category V	Category VI
Economic KPIs	For example • Revenue • Revenue related to product or product group • Revenue related to customer or customer group • Revenue due to special offer promotions • Revenue due to direct communication activities	For example • Profit • Profit margin • Customer profitability • Revenue due to discounts/costs in the form of lost revenues • Revenue due to participation in trade shows/cost of participation in trade shows

Fig. 3.17 KPIs for performance management in marketing and sales (Homburg and Krohmer 2006, p. 1234)

for sales and marketing. They differentiate between potential-related, market-related, and economic KPIs on the one hand, and effectiveness and efficiency-related KPIs on the other (see Fig. 3.17).

Although the above KPIs relate to marketing and sales, they are industry-independent, i.e., they are as relevant to the software industry as to any other. The

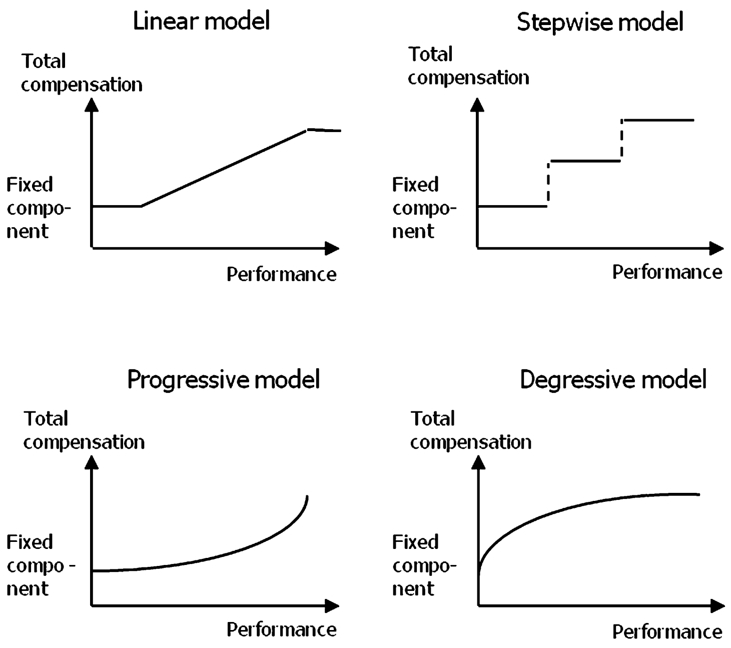


Fig. 3.18 Alternative performance-related pay models (Homburg and Krohmer 2006, p. 1266)

KPIs listed below, by contrast, have proven helpful for sales performance management in the software industry:

- Number of leads generated by direct marketing,
- Volume of addresses,
- Number of first visits,
- Opportunities weighted by closure probability,
- Number of solution presentations at customer,
- Number of proposals/quotations submitted,
- Quotation value,
- Number of contract negotiation meetings,
- Number of deal closures,
- Lost opportunities/unsuccessful quotations, and
- Customer visits per salesperson per month.

These KPIs can also be used for calculating variable, performance-based pay for sales staff. In line with principal-agent theory, the objective is to develop an incentive-compatible compensation model (see Sect. 2.5), i.e., design a model that motivates the agent (sales person) to pursue the goals of the organization (principal). A core feature of such a system is the division of pay into a fixed and a variable component. Figure 3.18 shows various basic alternatives.

The linear model is standard practice. The variable component should amount to about 30–40 % of the total target compensation. One of the key issues to be addressed is the selection of parameters used to determine the variable component.

In the software industry, sales targets or order inflow targets are most often used to determine the variable component. Concrete answers are needed to the following questions:

- What sales period will be used (usually the fiscal year)?
- What counts toward fulfillment of sales targets?

The second question can be highly contentious. The most delicate question for software companies is how maintenance business is handled. The significance of this question is apparent when we remember that maintenance income is generated over long periods and can account for up to 80 % of a software company's total annual sales (see Sect. 1.5).

During the sales process, setting the right price is a constant consideration, as it is crucial to a successful sales strategy. Against this background, the following section will investigate software providers' pricing strategies.

3.3 Pricing Strategies

3.3.1 Background

Pricing plays a key role in most organizations' strategies (Simon 1992, p. 7). It directly affects revenues and therefore, in the long term, profits. Incorrect decisions can jeopardize the company's reputation and customer relationships. Despite its importance, pricing strategies are often deficient in a number of respects, including lack of rationality in the shape of ad-hoc or arbitrary decisions (Florissen 2008, p. 85). Small and midsize enterprises are by no means the only ones to frequently rely on gut feeling when they make pricing decisions. But basing pricing strategies on empirical data can make a great deal of economic sense. Studies show that price adjustment at the right time usually has a greater impact on profit than a reduction in costs. For instance, a price adjustment of just 1 % can lead to a rise in operating profit of some 8 % (Marn et al. 2003).

However, conventional pricing models are not directly applicable to software products (Bontis and Chung 2000, p. 246). Of the many characteristics of software as a good (mentioned in Sect. 2.1), the fact that it can be duplicated virtually for nothing is particularly significant when it comes to setting a price. To recap: it is relatively expensive to develop a first copy of a digital good. But the marginal cost of an additional copy is near zero. How does this affect pricing? First of all, clearly, cost-based pricing must be ruled out. Demand- or value-based pricing makes far more sense. This means that software providers need to base their prices on how much their potential customers are willing to pay. Shapiro and Varian describe this relationship as follows: "cost-based pricing just doesn't work […].

You must price your information goods according to consumer value, not according to your production cost." (Shapiro and Varian 1998, p. 3).

In principle, the cost structure of digital goods lends itself to low-price strategies. These can be useful when, for instance, a software provider wants to squeeze out an established provider on a network market. Since the variable costs of the software product are negligible, the contribution margin would not be negative, even if the software were given away. This does not apply to physical products, as they are subject to variable costs. We will see below that the cost structure of digital goods also lends itself to price bundling, for example.

It would be a mistake, however, to assume that in economic and, specifically, in pricing terms software can be treated like any other digital good. While doing so might make sense for a software vendor that generates all or most of its revenues through license sales, it is by no means the general rule. In fact, consulting and support services do incur costs.

The following section provides an overview of the parameters that can be used in pricing software products.

3.3.2 Pricing Models for Software Products

3.3.2.1 Overview

Software products may be offered in many different forms. Over time, pricing models for software have changed fundamentally. Whereas during the mainframe era, prices were usually based on computing power, pricing models based on user numbers (licensing models) have been prevalent in the recent past (Bontis and Chung 2000, pp. 247–248). Today, software providers are increasingly offering usage-based pricing models, a topic we will examine in greater depth in Sect. 3.3.2.3.

Since there is no universally valid pricing model for software providers (Bontis and Chung 2000, p. 246) and pricing models can be comprised of multiple components, we will next describe various pricing parameters for software products. Figure 3.19 shows an overview of the parameters we will be discussing (Lehmann and Buxmann 2009).

Software providers' pricing models generally comprise a combination of parameters. The pricing model can include multiple sub-items from each column.

3.3.2.2 Price Formation

The provider defines how the price is to be formed. Both the pricing basis and the degree of customer interaction must be taken into consideration.

There are essentially three ways to determine prices (Homburg and Krohmer 2006, p. 720; Nieschlag et al. 2002, pp. 810–814):

- Based on cost,
- Based on value or demand, and
- Based on competition.

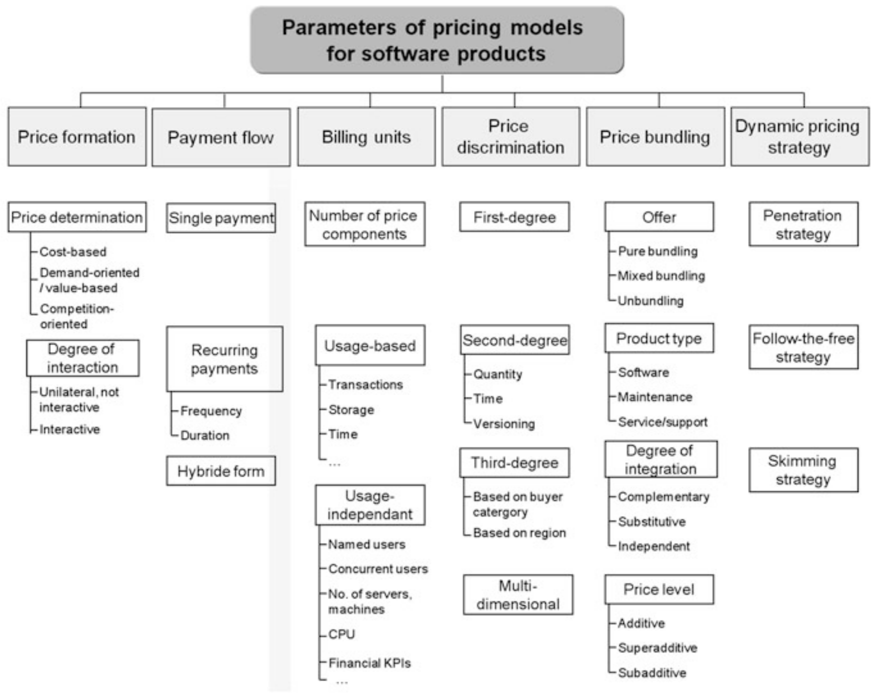


Fig. 3.19 Parameters of pricing models for software products

The cost-based approach determines the price using cost accounting methods (Diller 2008, pp. 310–311). However, this method of determining the price is of little significance when it comes to software licenses and digital goods in light of their unique cost structure. When pricing SaaS solutions, on the other hand, it can make sense to take costs into account.

Demand- or value-based pricing is oriented toward the level of demand for the product (Homburg and Krohmer 2006, pp. 720–721). Here, the significant factor is how much the customer values the product, rather than the product's cost (Harmon et al. 2005, p. 1).

Competition-based pricing takes into account the prices offered by competitors and their price-related behavior (Homburg and Krohmer 2006, p. 747). The attractiveness of a competing product to a customer partly depends on the homogeneity of the products and the structure of the market (Nieschlag et al. 2002, p. 813). In the software industry, network effects and the resulting customer lock-in effects make it essential for software providers to gain a large market share, especially when their product is very similar to the competitor's. Competition-based pricing therefore plays an important role for software products in addition to value-based pricing.

Another pricing parameter is the degree of interaction. Noninteractive price formation is when the provider sets the price unilaterally, without the customer having any say in the matter. Interactive pricing, in contrast, requires interaction between customer and provider. Examples of interactive pricing include negotiations and (online) auctions (e.g., Schmidt et al. 1998). But generally speaking, auctions of digital goods, including software, make little economic sense (Shapiro and Varian 1999, p. 23).

3.3.2.3 Structure of Payment Flow

There are essentially two types of payment flows for software: Customers can make a one-time payment and acquire the right to use the software for an unlimited period, or they make regular, recurring payments. The two types can also be combined (Kittlaus et al. 2004, S. 82).

The single payment option corresponds to the software licensing model widely used today. By purchasing a license, the customer normally acquires unrestricted usage rights.

Recurring payments can vary in frequency and duration. For example, customers can agree to pay monthly or annual subscription rates over a two-year period in order to use the software. These pricing models are primarily used for SaaS solutions (Cusumano 2007, p. 20), where customers use the provider's software via the Internet for the agreed payment period (this is sometimes referred to as a subscription or rental model; Buxmann et al. 2008b). This kind of pricing model benefits users because it enables them to employ software cost-effectively even for brief periods, since normally the monthly payments are substantially lower than a one-time payment for licenses (Cusumano 2007, p. 20). However, this customer advantage makes higher financial demands on the provider. For example, SaaS providers often find it hard to break into the black (Hill 2008, p. 48). According to a survey by SIIA et al. (2006, p. 5), US-based software providers expect that the subscription model will become more common than one-time payments in the form of license purchases (see also Sect. 3.3.3).

Hybrid payment models that combine one-time and regular payments are another option. For example, it is common to purchase a software license in conjunction with a software maintenance agreement. These usually specify annual payments amounting to a certain percentage of the (one-time) license payment. At present, many software providers charge a maintenance percentage of around 20 %. The advantage of this model for providers is that it generates relatively uniform payment flows. In Sect. 3.3.3, we will provide some empirical findings regarding forms of payment for SaaS solutions.

3.3.2.4 Billing Units

The choice of billing units is another way in which pricing models can be shaped. In other words, the price can be set per user or based on a time factor, for example. The billing unit plays a key role in determining whether the customer feels the provider's pricing model is *fair*.

First the provider determines how many components the pricing model consists of (Skiera 1999b). Each component is based on a pricing unit. For instance, the

Table 3.4 Examples of usage-based billing units (Lehmann and Buxmann 2009)

Billing unit	Description
Transaction	The price depends on the number of transactions completed using the software. Both transactions in the technical sense (e.g., Web service calls) and in the business sense (e.g., number of delivery items processed) can be applied
Storage	The price is measured in units of storage capacity utilized (e.g., per GB)
Time	The price is determined by the actual duration of use of the software (e.g., per minute)

Table 3.5 Examples of usage-independent pricing (units) (Lehmann and Buxmann 2009)

Billing unit	Description
Named user	Software usage rights are linked to specific persons and pricing is based on particular individuals
Concurrent user	This option enables simultaneous use of software by a predefined number of users
Server/machine	Customers are charged per server or machine. Software usage rights are linked to a server or machine
CPU	The software price is calculated by the number of CPUs it runs on
Master data	Pricing depends on the amount of master data maintained (e.g., customers, suppliers, employees, inventory items, rental units, land parcels, managed assets)
Sites	Prices are calculated by site. This can include special types of site (e.g., mines)
Production volume	Pricing is based on a metric of production (e. g. barrels of oil per day)
Key performance indicators	Pricing is based on Key Performance Indicators (e. g. revenue, expenses, budget)

pricing model can be divided into a fixed monthly charge and a usage-based component, such as storage capacity utilized. Skiera (1999b) showed that service providers can increase their profits considerably by using two pricing components rather than one.

As we indicated in the above example, billing units can be either usage-based or unrelated to actual usage of the software. In principle, many different billing units are conceivable. They can also be industry-specific, such as the number of rental units managed in the case of property management software. Table 3.4 includes examples of usage-based billing units.

Implementing usage-based pricing can result in fixed and variable administration costs, for instance, for monitoring usage and for billing.

Usage-independent billing units are not dependent on to what extent the software is actually used. See examples in Table 3.5.

From the software provider's perspective, one benefit of usage-independent pricing is that customers are generally prepared to pay more for unlimited usage

(Sundararajan 2004, p. 1661). Many customers overestimate how much they use (flat rate bias; Lambrecht and Skiera 2006, p. 221). Empirical findings on the prevalence of usage-based and/or usage-independent pricing models in the SaaS field are shown in Sect. 3.3.3.

Billing units are not only important for the customer, but also in terms of price discrimination. This will be discussed in more depth below.

3.3.2.5 Price Discrimination Strategies

Price discrimination means charging customers different prices for essentially the same product (e. g. Diller 2008, p. 227; Skiera and Spann 2000; Pepels 1998, p. 89). The goal of the provider is to capture more of the consumer surplus. In contrast to a pricing model with consistent prices, this can be achieved by figuring in varying degrees of consumer willingness to pay (WTP). Because different customers estimate the product's value differently, providers can differentiate their prices and achieve higher turnover (Diller 2008, p. 227). An example of this is an audio CD available in three separate versions with different price tags:

- The premium version contains a wide range of additional features, such as song booklets, a multimedia section enabling access to an exclusive Internet offering, or bonus tracks.
- The standard version offers the features of a conventional album.
- The basic version comprises simply the disk in a sleeve with no booklet.

Pigou (1929) distinguishes between first, second, and third degree price discrimination.

At first glance, the best strategy for software providers seems to be first *degree discrimination*. The idea behind this is to distinguish the price individually by offering the product exactly at the customer's reservation price (RP), i.e., the maximum amount the customer is willing to pay. Minimum price thresholds do not have to be considered in the case of digital goods because the variable costs are negligible. This type of pricing strategy is not a practical option for software providers in the business-to-consumer sector, of course, because they have millions of customers and it is impossible to make even a rough estimate of each customer's WTP. In contrast, vendors of standard business-to-business software are clearly perfecting the art of price discrimination. Prices in this segment are often complex and not very transparent. Price lists can be hundreds of pages long, and there is huge scope for negotiating contractual terms. This applies not only to software licenses but particularly to complementary consulting services. However, providers of standard software generally cannot afford too much in the way of price discrimination. For example, customers may well exchange notes in user groups and discover that others have paid a different price for the same software product.

Second degree price discrimination plays a major role in digital goods (Linde 2008, p. 209). It is based on the principle of self-selection, i.e., the customer selects its own product-price combination (Varian 1997, p. 193). Skiera (1999a, p. 287) makes a distinction among quantity-, time- and performance-based price discrimination with self-selection.[1]

In the case of quantity-based price discrimination, the average price per unit depends on the total amount purchased. Flat rate is included in this category too, since the average price per unit depends on the consumer's overall use (Skiera and Spann 2000). This type of volume discount is widespread for software licenses, especially for key accounts. Discounts in excess of 50 % are not uncommon.

Time-based price discrimination targets degrees of consumers' WTP at various times (Skiera and Spann 1998). An example of time-based price discrimination is not charging a fee to disclose stock market prices with a time delay, while charging for real-time prices. There are a host of other examples, such as seasonal pricing. One that applies to the software industry is pricing customer service according to the time of day.

Another type featuring self-selection is performance-based price discrimination. This is when relatively minor changes are made in service scope or quality (Diller 2008, p. 237) and the resulting product variants offered at different prices. In conjunction with product differentiation, this is frequently referred to as versioning (Varian 1997; Viswanathan and Anandalingam 2005).

Offering several versions of a product is seen as especially profitable for digital goods because of the cost structure (Viswanathan and Anandalingam 2005, p. 269). In the context of network markets, inexpensive variants can lead to greater market penetration. This shows that software products generally are well-suited to this type of price discrimination (Bhargava and Choudhary 2008, p. 1029). Software providers often develop a high-quality, feature-rich product to begin with, so they can then remove certain functionality and offer consumers different versions (Shapiro and Varian 1999, p. 63). Some examples of performance-based price discrimination include the Home Basic, Home Premium, Professional, and Ultimate versions of Microsoft Windows 7, which vary in functional scope and price.

It is important to bear in mind that too many different versions can be confusing for consumers and make more work for the provider (Viswanathan and Anandalingam 2005, p. 269). Because consumers exhibit extremeness aversion, the rule of thumb for information goods is to offer three versions so that the customer can compromise by selecting the mid-range variant (Varian 1997, p. 200; Simonson and Tversky 1992; Smith and Nagle 1995). The basic idea is that undecided consumers will tend to choose the product of medium quality. The following example illustrates this principle: If a fast food chain were to offer beverages in large and small sizes only, some customers with no clear preference would probably select the small size. But if the vendor also offered a jumbo size, and the new medium size were identical to what was previously the large size, many customers would choose the new medium size (Varian 1997, p. 199 f.).

Bhargava and Choudhary (2008) recommend that companies consider offering additional lower quality variants when variable costs are decreasing. The authors

[1] Because of its minor importance for the software industry, search-related price discrimination has been omitted.

give formal evidence that decreasing variable costs make versioning more profitable for providers because that way, they acquire additional customers with a lower WTP (Bhargava and Choudhary 2008, p. 1031).

Third degree price discrimination is based on how the provider segments the market (e.g., Diller 2008, p. 229). In contrast to second degree price discrimination, the consumer cannot self-select. There are two types: location-based and customer based (Skiera and Spann 2000).

The latter is often used to create a lock-in effect: "Although software producers don't hang around outside of schoolyards pushing their products (yet), the motivation is much the same." (Shapiro and Varian 1998, p. 46). For example, a software provider could give away its products to a specific group of consumers to achieve lock-in effects. The idea is that the pupils will learn to navigate the software and when they become paying consumers, will be more likely to buy the provider's product.

A simple kind of location-based discrimination is to sell products in different locations at different prices. Software licenses are sometimes priced this way, as are associated service agreements and consulting contracts.

Price discrimination that includes more than one dimension is called *multidimensional* (Skiera and Spann 2002, p. 279), and is very common practice. For example, pricing can be based both on location and on quantity. The provider has different prices for every country or region as well as a pricing model dependent on the quantity purchased. Multidimensional price discrimination can achieve a finer customer segmentation, which enables providers to exploit customers' existing WTP even more fully. But the complexity of the pricing model should be limited to avoid confusing the consumer, and to ensure that the provider's billing process remains feasible (Skiera and Spann 2002, p. 279).

3.3.2.6 Price Bundling

Price bundling is another parameter relevant to software pricing. In general, it consists of packaging multiple distinct offerings (products, services, and/or rights) from one or more providers and selling the package at a single price (Diller 2008, p. 240). It is sometimes considered a special form of price discrimination (Skiera et al. 2005, p. 290; Diller 2008, p. 240). Because of its significance in the software industry, we will devote a complete section to price bundling in relation to software pricing.

A variety of goals can be achieved through bundling. First and foremost, it can be used as a means of price discrimination (see Sect. 3.3.2.5) (Viswanathan and Anandalingam 2005, p. 264). While conventional methods of discrimination require comparatively detailed knowledge of each product's RP, this does not apply to bundling to the same extent (Adams and Yellen 1976, p. 476). Bakos and Brynjolfsson (1999) explain this with reference to the law of large numbers. According to this, it is simpler for the provider to estimate customers' WTP for a bundle that includes multiple products than for each product individually. This is because the WTP distribution for the bundle shows fewer extreme values

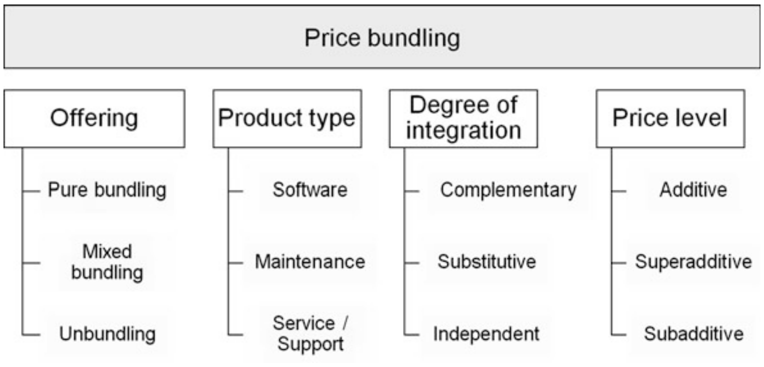


Fig. 3.20 Aspects of price bundling (Lehmann and Buxmann 2009)

(Viswanathan and Anandalingam 2005, p. 264). However, Wu et al. (2008, pp. 608–609) argue that this only applies when variable costs are zero. If there are any variable costs at all, even if they are extremely low, a bundle composed of numerous elements generates significant costs, which will lessen the potential advantages of bundling.

Because the software industry is heavily influenced by network effects, bundling can be advantageous to providers, as it drives the proliferation of (additional) products. For example, Adobe's Creative Suite not only includes the software for photo editing and layout design, but also the software for creating PDF files. As a result, sales of the Creative Suite also promote the spread of PDF documents. Moreover, a bundling strategy may impede market entry for potential competitors (Nalebuff 2004), e.g., for providers that offer only one of the products in the bundle. It can also reduce billing and shipping costs, as multiple products are sold in a single transaction (Viswanathan and Anandalingam 2005, p. 264; Adams and Yellen 1976, pp. 475–476).

We will explain the aspects of price bundling shown in Fig. 3.20 before discussing the factors that determine how advantageous bundling strategies will be.

Software providers' *offerings* can include pure and mixed bundling as well as unbundling. Pure bundling means that products are only offered as part of a package. If the customer can choose between purchasing the bundle or buying each product separately, this is referred to as mixed bundling. Unbundling is when the customer can only purchase the products separately (Adams and Yellen 1976; Schmalensee 1984, pp. 212, 475; Olderog and Skiera 2000, p. 140). Another option is customized bundling, in which the customer can choose, within specified limits, which products to include in the bundle. The provider determines merely the price and scope (Hitt and Chen 2005). Wu and Anandalingam (2002) show that offering multiple, customized bundles can be beneficial to a monopolistic provider of information goods. Assuming incomplete information, Wu et al. (2008) found that customized bundling is more profitable than unbundling.

The *product type* is another aspect of price bundling. The products offered together in the bundle can be very different in nature. In the software industry, the product types are usually the software itself, maintenance, support, and other services. Today, software vendors often generate revenue from three sources—licenses, maintenance, and other services—in equal measure (Cusumano 2007, p. 19). These three types of offering can be bundled in various ways.

Products in a bundle can also be described according to their *degree of integration*. Elements of a bundle can be complementary (Diller 2008, p. 241), substitutive, or independent of each other. Bakos and Brynjolfsson (1999) discovered that bundling a large number of unrelated information goods can be profitable. Their model also lends itself to analyzing complementary and substitutive elements of bundles. Stremersch and Tellis (2002) suggest that when the purpose of bundling is to add value for customers in comparison to using or consuming the products separately, it is better to use the term *product bundles* rather than price bundles.

Methods for determining bundle *pricing* can be additive, superadditive, or subadditive. In the first instance, the bundle price corresponds to the sum of the individual prices. Superadditive bundles are priced higher than the sum of the individual components' price tags, while subadditive bundles are priced lower than the sum (Diller 2008, pp. 240–241). The latter case, i.e., a bundle that offers a discount on individual prices, is considered the norm (Diller 2008, p. 241; Viswanathan and Anandalingam 2005, p. 264). A survey by Günther et al. (2007, p. 139) revealed that most respondents expected a lower overall price for the bundle when purchasing bundled Web services. For example, the price for the Microsoft Office bundle is significantly lower than the sum of the components' prices. When Microsoft distributed the Windows operating system bundled with Media Player, customers were under the impression that the latter product was thrown in for free. This type of bundling strategy can be used, among other things, to include new applications with older products in order to motivate customers to upgrade or opt for maintenance services (Cusumano 2007, p. 20). However, it is important to note that product bundling can contravene antitrust legislation. A prime example of this was the action taken by the European Commission against Microsoft for bundling its Windows operating system with Internet Explorer.

Olderog and Skiera (2000), among others, looked at which factors influence the benefits conferred by bundling, on the basis of a model by Schmalensee (1984). The success of bundling strategies depends primarily on two factors: first, the type and degree of correlation between the RPs, and second, the size of the variable costs compared to the RPs. These two factors will be discussed below. RPs are positively correlated when the consumers who would be prepared to pay a high (low) price for product A would also pay a high (low) price for product B. A negative correlation exists if the customers who would tend to pay a high (low) price for product A would be willing to pay a low (high) price for product B. In principle, the more negative the RP correlation, the more advantageous a bundling strategy will be for a provider. This is because a large negative correlation leads to lower variance in the bundle's RPs, creating a more homogeneous

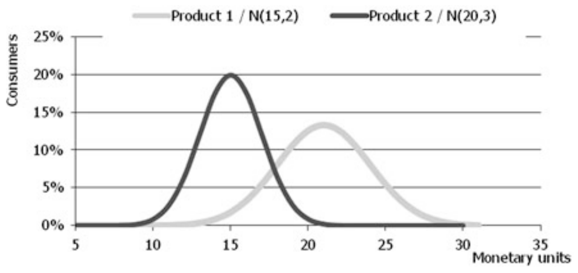


Fig. 3.21 Example distributions of reservation prices for two products (Olderog and Skiera 2000, p. 143)

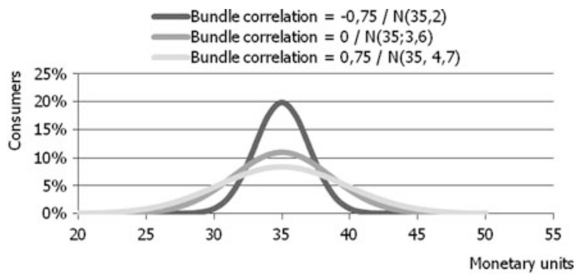


Fig. 3.22 Effect of RP correlations on the homogeneity of the RPs for the bundle (Olderog and Skiera 2000, p. 143)

demand structure (Olderog and Skiera 2000, p. 142). Figure 3.21 shows the RP distribution for two products and this relationship is depicted graphically.

Figure 3.22 shows how the correlation between the RPs for the individual products affects the homogeneity of the RPs for the bundle.

The homogeneity of the demand structure for the bundle increases (decreases), if the RP correlation is negative (positive). Figure 3.23 shows that the provider can increase the profit (dark area) when the demand structure is homogeneous.

As explained earlier, the success of a price bundling strategy also depends on the size of the variable costs in relation to consumer RPs (Olderog and Skiera 2000, p. 144; Bakos and Brynjolfsson 1999). We will now explain this using the simple numerical model shown in Table 3.6.

Here we are assuming that there are only two products, and comparing whether an unbundling or a bundling strategy is more successful when variable costs are relatively high. The first two lines give the maximum RPs of the two customers for products 1 and 2 and for the bundle comprising these two products. It is also clear that there is a negative correlation between the RPs of the two potential customers. The third line gives the optimum price for the two products and for the bundle. As the variable costs associated with making each of the two products are assumed to be seven monetary units, it is reasonable to set a price of eight monetary units for both products. This results in a positive profit contribution of one monetary unit each for products 1 and 2. Neglecting any fixed costs, the return generated by this bundling strategy is two monetary units. In light of this, it obviously makes no sense to pursue a bundling strategy as the variable costs are too high.

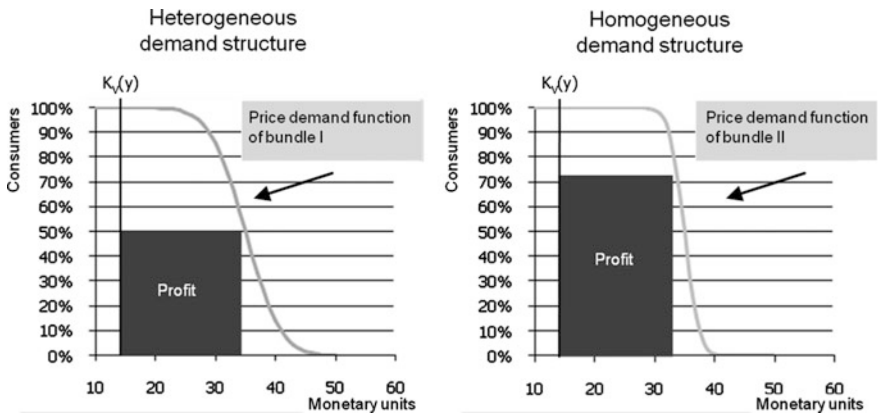


Fig. 3.23 Effect of a homogeneous demand structure on profit (Olderog and Skiera 2000, p. 144)

Table 3.6 Example of the effects of a bundling strategy when variable costs are relatively high

	Unbundling		Bundling
	Product 1	Product 2	Bundle
RP buyer $i = 1$	8	4	12
RP buyer $i = 2$	4	8	12
Variable costs	7	7	14
Optimal price	8	8	/
Profit contribution	1	1	/
Quantity sold	1	1	0
Profit	1	1	0
Total return	2>		0

But how does the picture change when the variable costs are lower? To show this, we recalculate our simple example with variable costs of zero (see Table 3.7).

As we can see, due to the homogeneous demand structure, a bundling strategy allows the provider to exploit the consumers' RPs to the full. So with lower variable costs, the bundling strategy is more successful than the unbundling strategy.

Finally, Fig. 3.24 shows how the variable cost level affects the success of bundling strategies.

For our numerical example, we have shown the profits for the bundling or unbundling options in accordance with the variable cost level. It is apparent that the critical level for variable costs is four monetary units. If they exceed this amount, unbundling is the more profitable alternative. Conversely, when the

Table 3.7 Example for the effects of a bundling strategy for digital goods

	Unbundling		Bundling
	Product 1	Product 2	Bundle
RP buyer $i = 1$	8	4	12
RP buyer $i = 2$	4	8	12
Variable costs	0	0	0
Optimal price	8	8	12
Profit contribution	8	8	12
Quantity sold	1	1	2
Profit	8	8	24
Total return	16<		24

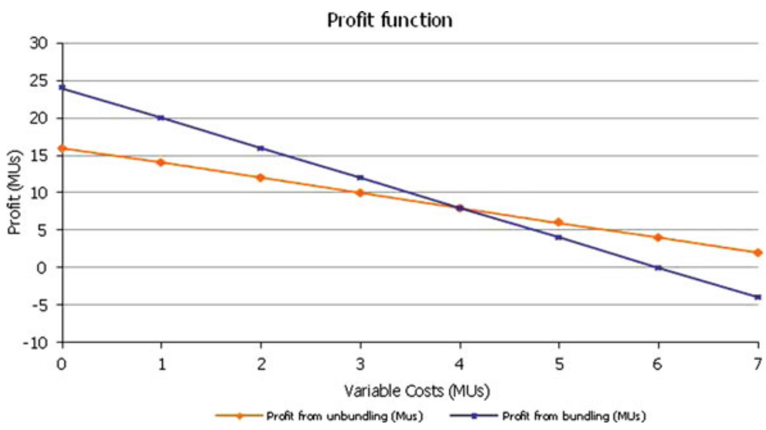


Fig. 3.24 Break-even point as a function of the variable costs

variable costs are lower than four monetary units, the provider gains more from bundling.

A further advantage of bundling for software providers is that it helps them to broaden the installed base of their products and thus generate network effects.

In conclusion, when there is a negative correlation between the RPs of the different products, bundling will tend to produce a more homogeneous demand structure. In addition, the lower the variable costs, the more advantageous bundling strategies become. For this reason, price bundling is a particularly useful strategy for software products. It should also be noted that the ideal number of products in a bundle can also depend on the presence of constraints on customers' budgets (Bakos and Brynjolfsson 1999).

3.3.2.7 Dynamic Pricing Strategies

Unlike the approaches we have examined so far, dynamic pricing strategies are based on a multiperiod horizon in which time prices will usually change. The following sections will focus on penetration, follow-the-free, and skimming strategies. We will not discuss pulsation strategies (where vendors alternately raise and lower the prices over time) as we cannot imagine any useful applications of them in the software industry.

The purpose of employing a *penetration strategy* is to quickly acquire market share by means of low prices. A low-price strategy can play an important role in network effect markets in light of the startup problem and lock-in effects, which were mentioned earlier.

The presence of network effects means that the software industry lends itself to penetration strategies. Furthermore, the type of cost structure in this sector, featuring negligible variable costs, favors low-price strategies. In fact, even giving away software will not lead to negative profit margins. In some situations, it may even be necessary to make a loss in order to catch up with a competitor who has a head start. However, whether this kind of strategy makes sense depends heavily on the network effect factor (i.e., the size of the network effects in relation to the total utility). The higher the network effect factor, which we discussed in Sect. 2.2, the more vital low-price strategies will be (Buxmann 2002). Furthermore, a penetration strategy will be especially worthwhile, if the provider is offering a product that is incompatible with the market standard.

A penetration strategy can include a second stage in which the provider increases prices once a critical mass has been reached. Ahtiala (2006) showed experimentally that given the problem of software piracy, it benefits software providers initially to sell their products at a very reasonable cost to generate a lock-in effect, and only to offer subsequent upgrades at a higher price. To a certain extent, this strategy has been observed in the real world: some companies—including big names—have given away licenses for free in recent years.

The *follow-the-free* strategy comprises two stages: first the products are given away to generate lock-in, then revenue is generated through the sale of complementary products or premium versions to the existing customer base (Zerdick et al. 1999, pp. 191–194). For instance, a company could offer its software product for free, and charge fees for associated services such as installation, maintenance, user training, and customizing (Cusumano 2007, p. 21). An example from the software industry is the strategy of Adobe, which succeeded in making its PDF format the industry standard by this method.

The pricing strategy of Adobe Systems Inc.

Adobe Systems Inc. is a leading provider of graphics, design, publishing, image, and video processing software for Web and print production. In 1993, the company set new standards with the development of the document description standard, PDF (Portable Document Format).

Adobe's success is mainly down to its unique pricing strategy: offering its Acrobat Reader free for viewing PDF documents, and selling complementary PDF creation software. In 2009, Adobe posted more than 20 % of sales with its Acrobat Software (Knowledge Worker segment). Today, the free Acrobat Reader has an adoption rate of around 89 %. Figure 3.25 shows the breakdown of Adobe's sales per product segment.

Adobe Flash Player

In 2005, Adobe Systems acquired its competitor Macromedia Inc., originator of the now widespread Flash technology. With this technology, Adobe pursued a pricing strategy similar to the one employed for its Acrobat product line: the Flash Player used to view Flash files is available as freeware. According to the company itself, the player is installed on about 98 % of Internet users' PCs worldwide. Customers who wish to create Flash components must purchase an authoring tool.

Through this complementary pricing strategy, i.e., distributing the viewing software free and selling the associated file creation programs, Adobe has succeeded in both cases to introduce a standard which is widely used and accepted, and to generate impressive sales.

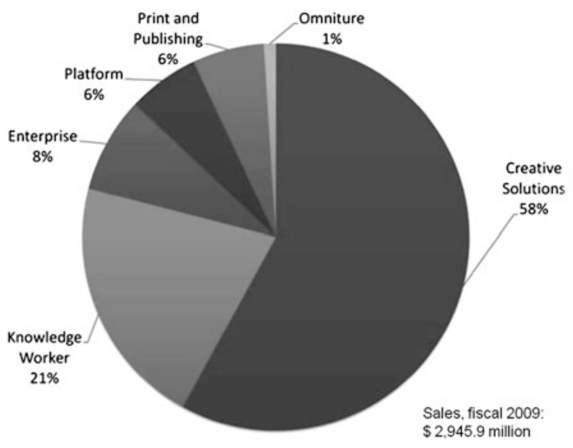
Adobe Systems Inc.

Adobe Systems Inc. was established in 1982 by John Warnock und Charles Geschke, the inventors of the PostScript document format. PostScript is a page description language used to describe the format of a printed page at the prepress stage. Building on PostScript's success, the company developed and launched the PDF standard in 1993. Since then, Adobe has published the source code. In January 2007, Adobe Systems submitted the PDF specification to the International Organization for Standardization (ISO), paving the way for it to become an official standard. The success of Adobe Systems Inc. is due in no small part to its innovative business strategy.

Sources www.adobe.com; Datamonitor (2006): Company Spotlight: Adobe Systems Incorporated, Financial Times Deutschland: http://www.ftd.de/technik/it_telekommunikation/133661.html.

In some cases, software products are offered on the basis of a *skimming strategy,* where the provider starts with a very high price and reduces it over time. The main purpose of this strategy is to exploit differences between consumers' RPs: in the high-price phase, consumers with a very high RP will acquire the

Fig. 3.25 Adobe's sales by product segment, 2009



product. Then, during the period in which the provider gradually reduces the price, it can exploit the RPs of the remaining consumers step by step. Computer games are a practical example of this strategy: very expensive when launched, they may even be distributed free of charge as magazine inserts at a later date.

3.3.3 Pricing Strategies of Software Providers: Empirical Findings

In this section, we will discuss how providers and users evaluate various pricing and licensing models. As few empirical findings on this subject have been published to date, most of the studies that exist include only a small number of parameters relating to software pricing.

Below we will discuss a study conducted in 2006 by the Software & Information Industry Association, which only addressed the parameter of billing units. We complement this with the results of expert interviews with users. The goal of our discussions is to cast more light on how users evaluate pricing, and why. However, it must be emphasized that the interviews are a qualitative rather than a representative sample of ERP system users.

The Software & Information Industry Association surveyed 698 experts. These comprised 487 software providers and 211 users (SIIA et al. 2006). Figure 3.26 gives an overview of the different licensing models, indicating which are preferred by software providers and users, respectively.

The most popular usage-independent billing unit among the users of software products is the number of concurrent users (see also Sect. 3.3.2.4.). Also attractive for users, but far less accepted, is licensing by servers or machines. Only a few customers favored billing by named users or processors. The least desirable form of billing is financial KPIs, preferred by only 1 % of users (SIIA et al. 2006, p. 7).

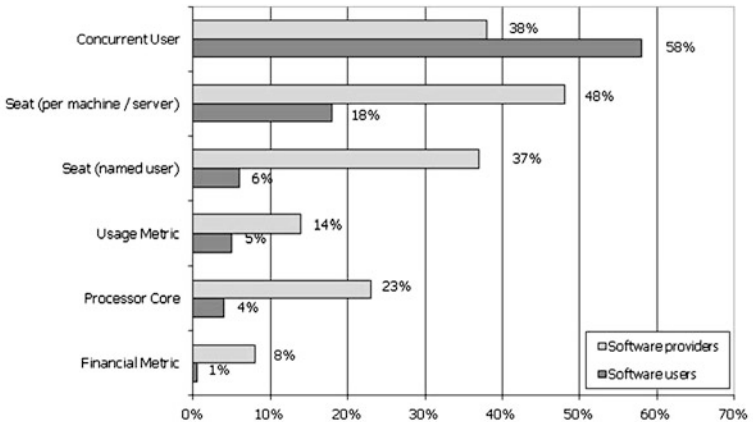


Fig. 3.26 The billing units preferred by software providers and users, respectively (SIIA et al. 2006, p. 7)

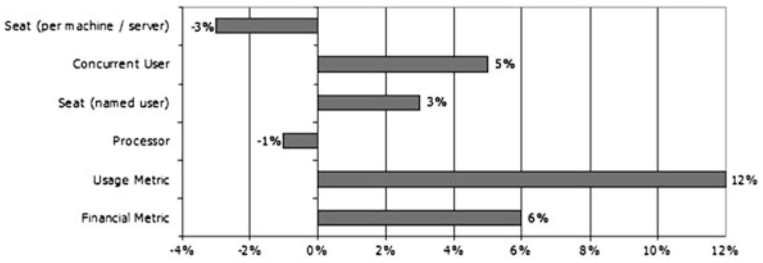


Fig. 3.27 How providers expect the use of billing units to change (SIIA et al. 2006, p. 7)

We will now look more closely at pricing models with usage-based billing units, as a key finding of the survey was that software providers expect this kind of model to become more widespread in future (see Fig. 3.27).

However, the findings also indicate that users are increasingly rejecting usage-based licensing models, as Fig. 3.28 illustrates

According to the SIIA survey, only one-fourth of customers were satisfied with the provider's pricing and licensing strategies.

We will next present the findings of a survey of experts conducted in 2008 to discover the reasons why users evaluate certain pricing models as attractive or unattractive.

Only three out of ten experts interviewed regarded usage-based pricing models for software as an attractive alternative. Seven expressed their skepticism or rejection of this pricing model. The two main reasons they cited were the difficulty of predicting costs, and the possibility that costs would fluctuate significantly due

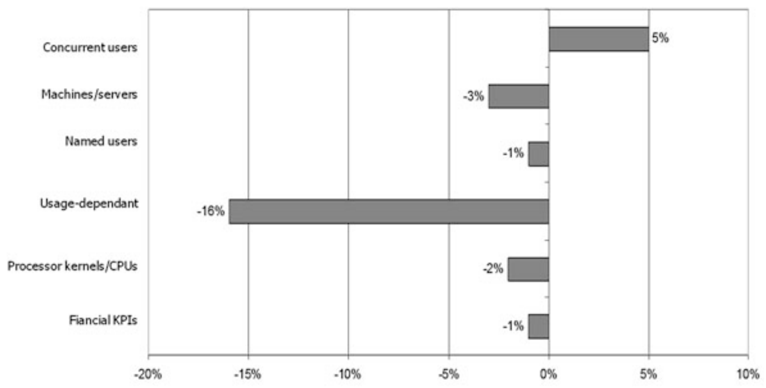


Fig. 3.28 Changes to users' billing unit preferences, 2005 compared with 2006 (SIIA et al. 2006, p. 7)

to varying usage levels. Difficulties lay not only in predicting costs, but also in determining the basis for billing. With a usage-based pricing model, it was essential that the effort needed to measure usage was not too high and that the model was straightforward and easy for users to understand. However, for some respondents, the structure of the model was consistent with their idea of fairness—these individuals declared themselves willing to pay more for using the software more (value-based pricing).

What survey participants did not like was the fact that most models force the user to pay for a certain level of usage in advance. Very often, additional fees are payable when actual usage exceeds the planned level. Conversely, users who do not fully exploit the planned usage level do not normally receive any refund or credit.

In general, respondents tended to look unfavorably at the pricing models currently used by software providers. This was due in part to the complexity of the models and the combination of multiple models. In this connection, respondents also expressed their dissatisfaction with IT outsourcing service providers. The latter often optimize their services on the basis of technical criteria, and this frequently leads to a mishmash of different licenses.

One respondent expressed his satisfaction with the conditions offered to large organizations. However, substantial discounts are only granted on licenses and not on maintenance fees. This has to do with accounting regulations. In the context of revenue recognition for multiple-element contracts (bundling), software vendors who prepare financial reports in accordance with US-GAAP are required to state a fair value, backed by objective evidence, for maintenance services yet to be provided under the terms of a contract. This is why these vendors do not offer discounts on maintenance services (Suermann 2006, pp. 112–114).

Asked what changes to pricing models user organizations would like to see, almost half of respondents expressed a wish for greater flexibility. They would like to be able to regularly adapt their agreements to changing user numbers, the option of usage for limited periods (including periods with no usage), and new billing units that are independent of user numbers. However, flexible pricing models can also be a disadvantage to customers, as providers can exploit flexibility to their own ends. This is particularly likely to happen with business (ERP) software, since changing to a different provider would entail significant switching costs for the customer. It is, therefore, unclear who profits most from more flexible pricing models.

With regard to pricing of SaaS solutions, respondents suggested usage–based billing (four votes) and a flat fee (five votes). They believed that these models offer the option of limited usage periods and reducing entry barriers, and in the case of flat fees, better cost planning. In addition, users would like to see a much reduced notice period, although one CIO pointed out the risks to the user organization of short notice periods.

3.3.4 Approaches to Pricing for Custom Software Providers

The approaches discussed above apply predominantly to standard software providers. For example, product and price differentiation methods and dynamic pricing are obviously far less relevant to custom software providers. We will now look at how providers of custom software determine prices, both in terms of opportunities and constraints.

Setting or estimating lower price limits is a good starting point. For this purpose, traditional cost estimation methods for software development projects can be used, such as COCOMO (II) and the Function Point method (Balzert 2000).

It is reasonable to use methods like these because custom software projects are very difficult to plan in a precise, reliable way. Project budget and schedule overruns are the norm, not the exception.

Cost estimation methods gage expected project effort and expense using certain parameters whose values are determined at the start of a development project. In its simplest form, COCOMO is based on an estimate of the number of lines of code. Project costs are estimated using tables based on empirical historical data. The Function Point method, by contrast—the method most commonly used to estimate the cost of software development projects—starts by evaluating project complexity in terms of parameters such as external inputs and outputs, user transactions, interfaces with external data sources, or the number of files used. Various inputs can be used to determine these quantities, such as specifications, entity-relationship diagrams, use case diagrams, screen layouts, and other documentation. Based on an evaluation of these criteria, tables are used to estimate the function points. A function point is an indicator of the scope and complexity of a proposed software solution.

ajor providers of custom software employ methods such as these, whereas smaller-scale software companies are more likely to rely on instinct and experience. One advantage of the methods is that they are relatively easy to use. On the other hand, they are also based ultimately on experience and empirical relationships, and some of their parameters are rather subjective. However, applying these methods can give providers valuable additional information for their decision making, particularly when it comes to major contracts. It often makes sense to apply these methods, given that this costs relatively little in comparison to the overall project costs. The methods deliver an estimate of project effort and expense, for example in terms of person-months, and of the likely development period.

The most significant source of expense for the software provider is the cost of the person-months invested. So, to determine a minimum price threshold for a development project, it makes sense to start by calculating the labor costs from the person-months. Labor includes both in-house employees and freelancers who are usually deployed during peak times. Labor costs for salaried employees comprise both fixed and variable pay components, employer contributions to social security, and training costs. Freelancers may be paid by hour or day, or may receive a flat rate for a given project or sub-project.

If the project effort expressed in person-months is weighted according to the costs of in-house versus freelance staff, this method will give a rough idea of the project costs, but no more than that. Manufacturers also use overhead costing and factor in indirect costs to determine the cost of making a given product. Although the same principle can be applied in the software industry, it is not generally necessary, because indirect costs—for offices, the use of software development environments, etc.—play a minor role in comparison to the labor costs.

Once these costs, with or without overhead costs, have been determined, the “only” thing left to do is to specify the mark-up. A large number of considerations must be brought into play, including the fact that the Function Point method tends to underestimate the actual costs.

The method described here is suitable for a cost-driven approach to pricing. Obviously, the prices determined this way will not necessarily be aligned with competitive demands or customers’ RPs. Against this background, other authors have suggested that it would be more appropriate to focus on the market situation when setting prices. However, this is extremely difficult to do when it comes to project business. After all, the unique, one-of-a-kind nature of projects makes it all but impossible to put a market price on them.

Psychological factors should also be taken into account. The relatively new discipline of behavioral pricing offers valuable insight here: it investigates potential customers’ attitudes to prices and pricing information, how they respond to prices offered, and how they use price information when evaluating products and making choices (Homburg and Koschate 2005). Some of its findings differ from the assumptions of classical price theory. For example, people often evaluate prices relative to a reference value, rather than in absolute terms: they often cannot recall prices; or they abandon a price search half-way through. Behavioral pricing primarily pursues a descriptive research approach and focuses mainly on cognitive processes not discussed by classical price theory.

Even though a cost-based pricing strategy has the drawbacks described above, it can still provide a good basis for determining whether a given project can be performed in a cost-effective way, at least within a short-term framework. But it generally makes sense to base decisions on other objectives, too, such as opportunities for acquiring key customers, defending and extending market share or preventing competitors from penetrating the market.

Having devoted the preceding two sections to market-driven strategies, we will now turn our attention to development strategies.

3.4 Development Strategies

Academics and industry professionals have long concerned themselves with the efficient and effective development of software; previously, this was primarily considered in the context of user organizations, but is now increasingly analyzed from the perspective of software providers. A myriad of approaches have been proposed, tested, and discarded. Over the following pages, we will provide an overview of the most important approaches to have survived the test of time.

3.4.1 Structuring of the Software Development Process

The structuring of the development process is pivotal to the development of software. After a brief description of the early days of software development, we will discuss plan-based approaches. Agile development, which arose as a backlash to this approach, forms our next subject. Finally, we will compare the merits of the various approaches in conjunction with the relevant contextual parameters.

3.4.1.1 Ad-hoc Development

Along with the first program codes, one of the first software development methodologies also came into being in the 1950s and early 1960s. Under this methodology, also known as the "stagewise" model, software is developed in strictly sequential stages. In general, however, the development of software at this time more closely resembled a craft (hence the term "software crafting"), which depended on the abilities of the individual developers (Boehm 1986, p. 22; Dogs and Klimmer 2005, p. 15).

Software was generally developed according to the Code and Fix approach. This was not really a methodology as such. It simply meant developing functionality without much advance planning and continuing to make modifications to address any errors until the program ran without any error messages. The inevitable outcome of this process was highly unstructured and a hard-to-maintain program code: the infamous spaghetti code (Boehm 2006, p. 13).

3.4.1.2 Plan-Based Approach

The increasing popularity of computers also raised the demands on the associated software. As software became more and more unwieldy, cost, time, and quality goals were often unachievable. As a result of this “software crisis”, the term “software engineering” was coined at a conference in 1968. Modeling software development more closely on the structured processes prevalent in engineering was intended to help overcome the software crisis (Dogs and Klimmer 2005, p. 16 ff.). Consequently, 1970 saw the invention of the “waterfall model” of software development. This built on the stagewise model mentioned above, and was regarded from then on as the archetype of plan-based development methodologies (Boehm 1986, p. 22).

The defining characteristic of plan-based methodologies is that software development is carried out in a series of standardized, usually sequential steps. Based on initial requirements elicitation and detailed planning, a range of artifacts is created in each stage, such as user requirements documents or technical designs. These form the basis for the next stages. After the actual programming, the software is tested for correct implementation.

However, in the ensuing years the waterfall model and plan-based approaches in general, were criticized. One criticism was that the sequential process made it difficult to accommodate changes to the customer requirements during the development process. This necessitated extensive, laborious modifications to the artifacts. The use of new methodologies, such as rapid prototyping, results in new approaches, including the spiral model (Boehm 1986, p. 21 ff.) and the V model (Hindel et al. 2004, p. 16), but none of this altered the fundamental limitations of the plan-based methodologies.

Apart from the term software engineering gaining widespread acceptance, none of the various approaches achieved the sought-after success. The old problems, cost blow-outs, lengthy project times, and poor quality, continued to exist. Numerous reasons have been put forward for this, including the following (Dogs and Klimmer 2005, p. 19 ff.):

- Plans quickly become out-of-date, as new customer requirements arise constantly.
- There are too many overheads (such as specifications and designs) which are irrelevant to the goal of developing high-quality software for the customer.
- Too little attention is paid to the human factor during the development process.
- The first tests take place too late on in the software development process.
- Not enough is learned from mistakes to benefit future projects.

In addition, increasing globalization and a more dynamic business environment are creating ever more volatile requirements. Critics allege that the inflexibility of plan-based approaches frequently means that software is developed which, by the time it is finished, the customer no longer needs.

3.4.1.3 Agile Approach

As a result of the dissatisfaction with plan-based approaches, a variety of “lightweight” approaches were developed in the early 1990s, independently of one another. The term “lightweight” is intended to convey a contrast with the previously common "heavyweight" methodologies (in the sense of being process- and document-heavy) and enable a team to react as flexibly as possible to changes to the user requirements. Examples of this approach include Scrum, extreme Programming (XP), and Adaptive Software Development (ASD).

Lightweight methods follow a "just enough" approach. They attempt to eliminate processes that contribute minimal added value to the final software product. To this end, any artifacts that are created, such as specifications, are pared back to bare essentials, to keep the time and trouble of adjustment to a minimum in the event of changes. A key aim of lightweight approaches is to provide customers with the desired functionality incrementally (building on achievements to date)—as prioritized by the customer—in short, iterative development cycles. In addition, software tests do not simply occur at the end of projects, but are integrated into the development process.

Instead of seeing the constant changes to user requirements as a negative influence, lightweight methodologies see this as an opportunity to develop innovative software and so add value for customers. The role of developers is also different: Instead of extensive documentation, lightweight methodologies emphasize extensive collaboration among developers, including exchanging knowledge and experience. This means that the human factor plays a larger role than with plan-based approaches.

In February 2001, representatives of various lightweight methodologies met to establish commonalities and reach a joint understanding. One outcome was that participants agreed to use the term "agile" instead of lightweight, as the latter has negative connotations. In addition, they adopted the so-called "Agile Manifesto," which was signed by all attendees and which sets forth their joint understanding with regard to agile methodologies (Fowler and Highsmith 2001, p. 28 ff.).

The values set out in the Agile Manifesto represent a shift in the importance placed on various aspects of the software development process. In the words of Fowler and Highsmith (2001, p. 29 ff.):

"We value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan."

In addition to these four values, the Agile Manifesto also defines 12 principles around the values. But it deliberately avoids prescribing concrete programming practices. Rather, the principles are couched in sufficiently general terms, so that users can follow them yet still have sufficient creative freedom.

Critics allege that the principles underpinning agile development are neither new, nor do they represent a paradigm shift. In fact, the origins of iterative and incremental development can be traced back to the late 1960s and early 1970s (Fitzgerald et al. 2006, p. 200; Larman and Basili 2003, p. 48). What is new is the way the various agile methodologies interact, mutually reinforcing each other. In addition, proponents of agile methodologies recommend implementing

Table 3.8 Comparison of plan-based and agile software development approaches

Aspect	Plan-based	Agile
Attitude to change	Disturbance	Opportunity
Process	Sequential	Iterative
Artifacts	Comprehensive	As many as necessary
Role of developer	Resource	Decision maker
Releases	Full	Incremental

Table 3.9 Factors influencing the selection of methodology

Factor	Plan-based	Agile
User requirements	Stable	Unstable
Development team size	Tends to be large	Tends to be small
Security-critical systems	Suitable	Unsuitable
Type of software developed	Standard software	Custom software

iterations in a comprehensive way, to generate code and new software versions as fast as possible. Moreover, they advocate continuously modifying rough plans—providing a flexible way of dealing with problems and new requirements, which spring from the increasing speed of change in the business and technology worlds.

In any case, agile methodologies' basic principles are fundamentally at odds with those of plan-based software development, especially the advance preparation of plans in as much detail as possible, working in clearly identifiable phases and producing comprehensive documentation. The apparently irreconcilable differences between the two methodologies led to an increasing polarization of views, which was also known as the "Method War." This episode involved agile advocates being labeled "hackers," while supporters of plan-based approaches were likened to "dinosaurs" by their opponents. (Boehm and Turner 2003, p. xiii; Boehm 2002, p. 3; Beck and Boehm 2003, p. 45).

Table 3.8 provides an overview of the differences between plan-based and agile approaches in several key aspects.

However, the literature increasingly stresses that both agile and plan-based methodologies have their merits, with strengths in different areas. Neither methodology can be seen as fundamentally superior to the other (Boehm and Turner, pp. 148ff.). Table 3.9 shows which methodology appears to be best suited for which task in which application, on the basis of selected variables.

More recent approaches combine both plan-based and agile elements to exploit the strengths of both methodologies. For example, these hybrid methodologies plan the interaction of individual software components in great detail. The development of the components themselves, however, follows an agile approach.

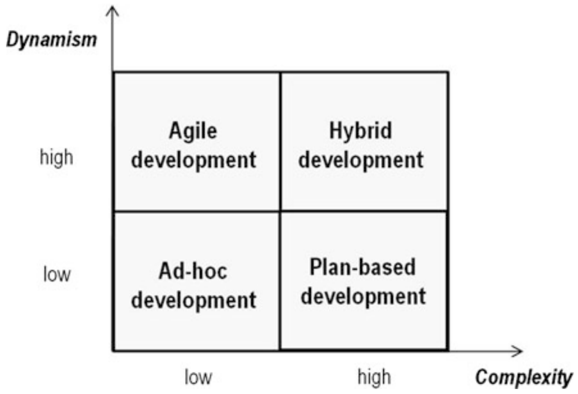


Fig. 3.29 Advantages of the different development methodologies

Building on the issues discussed in this section, Fig. 3.29 illustrates which development methodology is generally best suited in which context, in the form of a dynamism and complexity matrix. The *dynamism* axis represents an environment with increasing economic or technological uncertainty, in which user requirements frequently change. *Complexity* includes aspects such as the number of software developers respectively, the size of the team involved or the need to integrate software and hardware from third parties.

Of course, this depiction is only a starting point for selecting an approach, as other factors also play an important role within a given organization, such as experience with a methodology or the specific type of the software being developed.

3.4.2 Software-Supported Software Development

The first attempts to generate software code automatically, based on models, took place in the 1980 and 1990. This was known as Computer Aided Software Engineering (CASE), and in retrospect, was only moderately successful. CASE approaches had a fundamental weakness (Schmidt 2006): their closed nature. CASE tools generate code for a single specific target environment. Moreover, the approach requires that all stages of software development are supported by just one tool, and executed sequentially in the traditional way.

New approaches, Model-Driven Engineering (MDE), address these two weak points. We are deliberately using this term to stress the development process (engineering). In the literature, the term Model-Driven Architecture is frequently used, which shifts the focus onto the architecture of the software creation tools. Essentially, MDE is based on models and transformation processes (Petrasch and Meimberg 2006):

- Models are used to represent application domains. There are two types, Platform Independent Models (PIMs) and Platform Specific Models (PSMs).

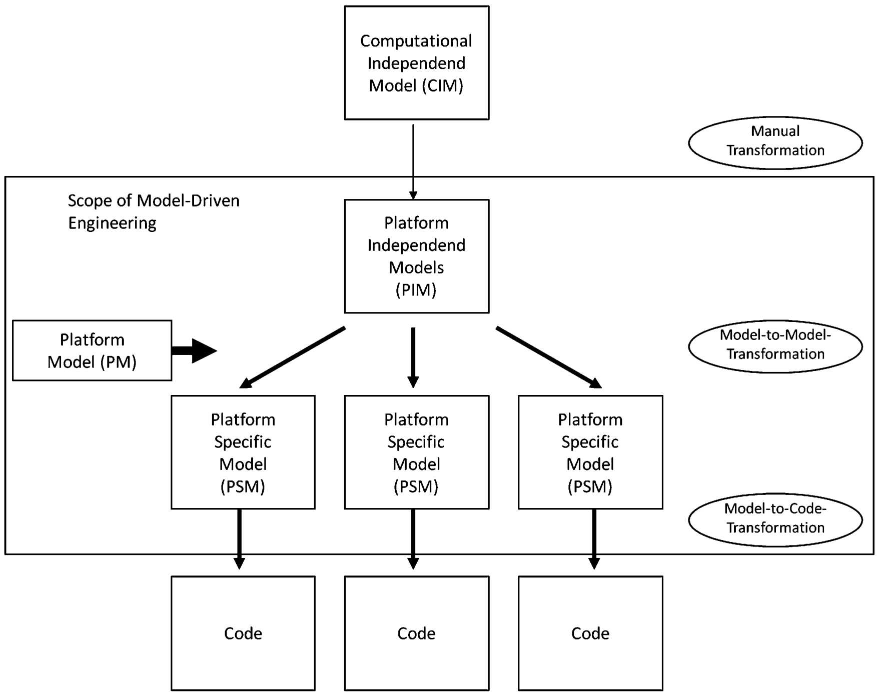

Computational Independend Model (CIM)
Manual Transformation
Scope of Model-Driven Engineering
Platform Independend Models (PIM)
Platform Model (PM)
Model-to-Model-Transformation
Platform Specific Model (PSM)
Platform Specific Model (PSM)
Platform Specific Model (PSM)
Model-to-Code-Transformation
Code
Code
Code

Fig. 3.30 Interaction of key concepts of Model-Driven-Engineering

- Transformation processes describe the automated mapping of language constructs of PIMs to PSMs (model-to-model transformation) and of PSMs to executable code (Model-to-code transformation).

Figure 3.30 shows how these two central MDE concepts interact, as well as the Computation Independent Model (CIM), which describes the domain to be supported by the software in informal, non-technical terms, and serves as a basis for the PIM.

Tools to support MDE typically include model editors, transformation editors, and tools and a repository. XML-based interfaces are available that allow different tools to exchange models.

The implementation of MDE approaches is still in its infancy. With respect to development costs, it can be assumed that the introduction and optimization of MDE approaches will give rise to non-project-related costs, while project-related costs in the later stages of development will be reduced. In other words: Implementing MDE only makes sense from the cost perspective, if

- the costs saved in development are not offset by additional costs for more extensive modeling and for the MDE implementation itself; and moreover
- a critical mass of projects is achieved.

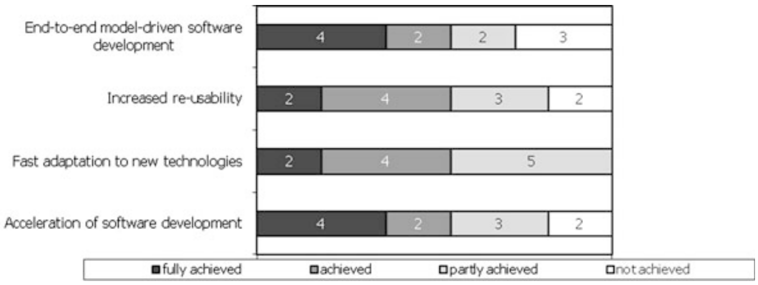


Fig. 3.31 Reported effects of MDE

MDE is a management concept that is yet to prove its effectiveness in the software industry. For this reason, we sent a questionnaire to 240 software providers in Germany asking for their view of selected aspects of this industrialization concept. 25 replied (Hess et al. 2007). This sample is small and not representative of the industry as a whole. However, it allows some initial conclusions to be drawn, which are presented below.

A significant majority of respondents were familiar with the concept of MDE, although it was more widely known in larger enterprises (annual net sales over 10 million euros) than in smaller companies. Almost half of the respondents already had some experiences with the MDE concept. These experiences varied considerably, as Fig. 3.31 shows. Nevertheless, there are some companies that successfully deployed the concept. In particular, they reported advantages in relation to end-to-end model-driven and accelerated software development.

More than half of the respondents expect MDE to play an important role in future. This assessment was more or less shared by the companies that had some experience of MDE. In summary, this suggests that MDE is a means for supporting software development that is seen by businesses in a positive light, but without euphoria.

3.4.3 HR Management in Software Development

Not least with the rise of agile development methodologies (see Sect. 3.4.1), it is increasingly accepted that the human factor plays a critical role in the software development process. The influence of such "soft" factors can play a greater role than technical factors as the type of technology or the use of a particular tool. The People Capability Maturity Model (CMM), developed by the Software Engineering Institute, addresses this issue. It applies the CMMI maturity model for assessing development processes to a company's HR management (SEI 2010).

Having noted the importance of effective HR management for the success of software development projects, the following section looks in greater detail into the question of whether software developers have special characteristics that set

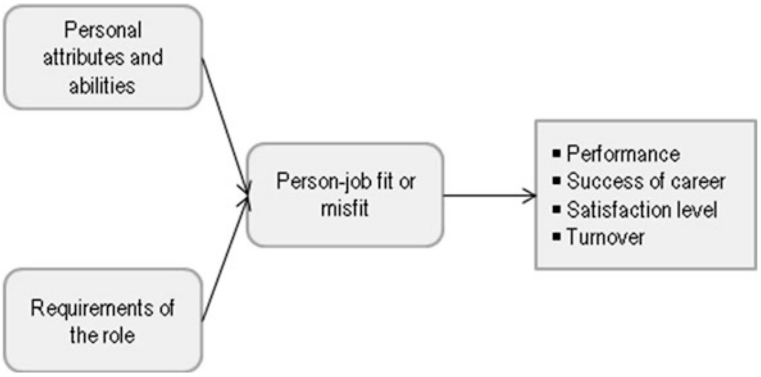


Fig. 3.32 Person-job fit (based on Lauver and Kristof-Brown 2001)

them apart from other professionals. Further to this, we will look at the "person-job fit" approach as a possible method to assign employees to roles and responsibilities. Finally, we will discuss the issue of employee motivation.

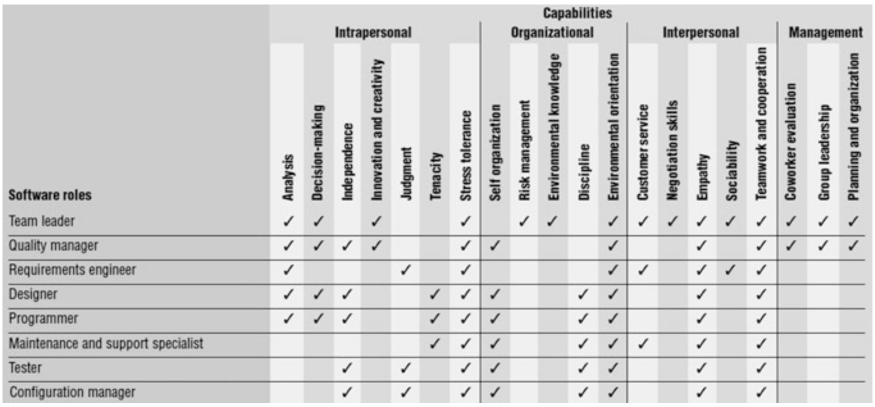
The ability to determine and even measure personality traits and types has caused increasing scientific interest in personality theories in the last few decades. Given that a person's attitudes, beliefs, perceptions, and behavior are all influenced to a certain degree by their personality, the personality structure has a considerable impact on software development processes.

"Personality" refers to an individual's unique psychological attributes, which, in turn, shape a multitude of characteristic behavior patterns. Consequently, personality traits can at least partially explain people's interest in particular careers (Costa et al. 1984). Software developers generally possess a different personality type to that of the average population. They tend to be less dominated by their emotions, are more likely to be introverted and in many cases would rather work alone than as a team member (Capretz 2003).

Despite these similarities between software developers, there are of course considerable differences within this group; it would be wrong to talk of a "typical" software developer. Moreover, a software development project incorporates a variety of roles (e.g. team leader, quality manager, tester, programmer etc.), which have different demands.

The "person-job fit" approach focuses on assigning people to particular roles based on their specific abilities. The model, illustrated in Fig. 3.32, posits that an individual's personal attributes and skills need to correspond with the requirements of their job. A closer fit generally leads to employees performing better, enjoying greater career success, and experiencing higher satisfaction levels, as well as less staff turnover.

Applying this to the software industry, Acuña et al. (2006) identify certain capabilities that are typical for people with particular personality traits. In the second step, these capabilities are matched against the specific requirements of eight different role profiles found in a software development project (see Fig. 3.33).



	Capabilities																			
	Intrapersonal							Organizational					Interpersonal					Management		
Software roles	Analysis	Decision-making	Independence	Innovation and creativity	Judgment	Tenacity	Stress tolerance	Self organization	Risk management	Environmental knowledge	Discipline	Environmental orientation	Customer service	Negotiation skills	Empathy	Sociability	Teamwork and cooperation	Coworker evaluation	Group leadership	Planning and organization
Team leader	✓	✓		✓			✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
Quality manager	✓	✓	✓	✓			✓	✓				✓			✓		✓	✓	✓	✓
Requirements engineer	✓				✓		✓					✓	✓		✓	✓	✓			
Designer	✓	✓	✓			✓	✓	✓			✓	✓			✓		✓			
Programmer	✓	✓	✓			✓	✓	✓			✓	✓			✓		✓			
Maintenance and support specialist						✓	✓	✓			✓	✓	✓		✓		✓			
Tester			✓		✓		✓	✓			✓	✓			✓		✓			
Configuration manager			✓		✓		✓	✓			✓	✓			✓		✓			

Fig. 3.33 Matching developer roles with requirement profiles (Acuña et al. 2006, p. 98)

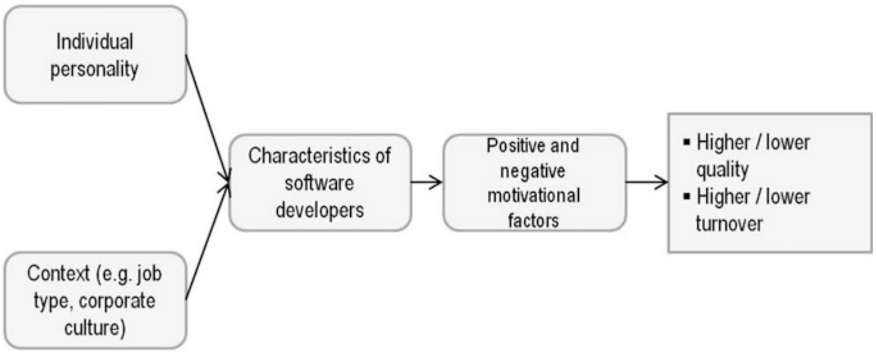

Individual personality
Context (e.g. job type, corporate culture)
Characteristics of software developers
Positive and negative motivational factors
▪ Higher / lower quality
▪ Higher / lower turnover

Fig. 3.34 Motivation of software developers (own illustration, based on Beecham et al. 2008)

Personality tests may also be used to determine whether an employee is suitable for a particular role. The capabilities identified in the test are then matched up with the role profile. It should be noted that this matching process provides only initial guidance as to whether an employee is suitable for a role—this model cannot take into account employees' specific skills or experience, the particular requirements of the position, or organizational details.

Closely related to the assignment of people to roles and responsibilities is the issue of how best to motivate software developers. However, it is difficult to provide a general answer, as positive and negative motivational factors depend on the specific context and the person's individual needs. Figure 3.34 offers a general model for explaining software developers' motivation.

Although there are many unique, context-specific factors, the literature emphasizes identification with the task as the primary motivator for software developers: in addition to personal interest in the task, developers are motivated by clearly defined targets and an appreciation of the significance of the task for the project as a whole. Other incentives include the existence of clear career paths and a varied and challenging range of activities (cf. Beecham et al. 2008).

A poor working environment, for example, lacking in key resources such as hardware and software, is commonly cited as a negative motivator. Others include poor management, such as calling superfluous meetings, and poor pay (cf. Beecham et al. 2008).

When implementing measures to increase motivation, attention should be paid to the previously discussed fit between the task and the person in question, as a misfit is a negative motivation. In other words, assigning the "wrong" person to the job can only be corrected to a limited extent by other means.