

Having described the basic rules of the game in [Chap. 1](#), we now turn to the economic principles in the software industry. We begin by discussing the properties of digital goods ([Sect. 2.1](#)), and go on to examine network effects on software markets ([Sect. 2.2.](#)) and the closely related issue of standardization ([Sect. 2.3](#)). We will also be looking at aspects of transaction cost ([Sect. 2.4](#)) and principal-agent theory ([Sect. 2.5](#)) that are of particular relevance to the software industry.

2.1 Properties of Digital Goods

Early in [Chap. 1](#), we introduced some of the economic properties of digital goods and software products. We will now discuss these in greater depth. A key property of digital goods is that producing the first copy generally incurs high costs, but that subsequent copies can be made at very low variable costs. Let us take a closer look at this phenomenon as it relates to the software industry. The development of software (or to be precise, the source code as the first copy), usually requires considerable investment, but there is no guarantee that a development project will be a success. For this reason, a vendor of custom software will be keen to pass on at least some of the development costs and associated risk to the customer (compare [Sect. 2.5](#)). A standard software vendor does not have this option, but bears the whole risk. At the same time, it can make a substantial profit if the product is popular. In addition, development costs are sunk costs, i.e., costs that have already been incurred and as such can no longer be influenced, so they are no more decision relevant.

However, we must point out that by assuming that variable costs are close to zero, the theory of digital goods considerably simplifies, and in many cases, oversimplifies, the real state of affairs. Only the variable costs of software licenses are close to zero. However, what are not negligible are the variable costs for providing services associated with the software, such as consulting, maintenance, and support. We will return to this issue in [Chap. 3](#).

Another important property of digital goods is that they can be copied easily without loss of quality. This explains why the costs of reproducing software are so low. Copies of digital goods are termed perfect, since the original and the copy do not differ.

The resulting problems are well known, for example, in the music business. Various peer-to-peer platforms enable users to download their own free (and often illegal) copies of music files. The same problem affects the software industry, particularly vendors whose products require little or no modification before use, and which are popular with noncommercial users. For example, office applications and computer games are often copied illegally and exchanged on file sharing networks. Several studies have attempted to estimate the loss of revenue suffered by software companies through digital piracy, including the Piracy Study published annually by the business software alliance (BSA). However, these studies have been criticized for their methodologies and assumptions. Many of them simply assume that every illegal copy leads to a direct loss of revenue, i.e., that every owner of a pirate copy would otherwise have purchased the software product. This results in immense (theoretical) losses that seem rather unrealistic, although there is no doubt that piracy costs software providers dear.

To protect their intellectual property and/or prevent illegal copying, vendors of digital goods can use digital rights management systems (Hess and Ünlü 2004). These provide protective methods based on hardware and software, including controls on access to, and use of, the software, protection of authenticity and integrity, identification via metadata, the use of copy protection, or a specific payment system.

2.2 Network Effects on Software Markets: The Winner Takes it All

In this section, we will discuss network effects, which play a significant role on software markets. The reason for this is that in addition to the functionality of a software product, especially its current and future distribution has a decided influence on its usefulness for users. This relationship is described and analyzed within the context of the theory of positive network effects. The first part of this section provides basic background information and definitions (Sect. 2.2.1). After that, we will explain why it is not always the best standards and the best software solutions that come to dominate markets with network effects, and why in many cases small software companies and startups tend to struggle (Sect. 2.2.2). First, we will describe these problems with reference to the so-called penguin effect. We will then present a model proposed by Arthur that illustrates how small chance events can determine the success of standards and software solutions in markets with network effects. We will go on to discuss why “winner takes all” so often applies to this kind of market, and how this ultimately explains why acquisitions are so frequent in the software industry (Sect. 2.2.3). Following on from this point,

we will show how software providers can exploit the existence of network effects to enhance their competitiveness (Sect. 2.2.4). Section 2.2.4 will explain both platform strategies and two-sided network effects with reference to the digital game industry. Section 2.2 concludes with a discussion of the limitations and potential extensions of network effect theory (Sect. 2.2.6).

2.2.1 Network Effects: Basics and Definitions

Michael Katz and Carl Shapiro define network effects as follows: “The utility that a given user derives from the good depends upon the number of other users who are in the same network as he or she” (Katz and Shapiro 1985, p. 424). In other words, the larger the network, the more the users benefit. Network effects are either direct or indirect.

Direct network effects arise from the fact that by employing the same software standards or common technologies, users can communicate with each other more simply and therefore more cost-effectively. The classic example of a direct network effect is the telephone: the more people own one, the more beneficial this technology becomes for users. The same principle applies to XML vocabularies such as xCBL, which are more useful the greater the number of organizations employing them. Inter-enterprise network effects also play a growing role in the context of ERP systems. The use of standard formats, for example, simplifies the exchange of business documents between different ERP systems. For this reason, stronger value-chain partners, for instance, in the automotive sector often pressurize smaller companies into deploying a compatible or identical ERP system to their own. Intercompany process standardization takes this approach a step further.

Indirect network effects, by contrast, result from the dependency between consumption of a basic good and consumption of complementary goods and services. They arise, therefore, when the wider adoption of a good generates a broader range of associated goods and services, which enhances the utility of the basic good. Indirect network effects occur in the context of standard software and complementary consulting services, operating systems with compatible application software, or with regard to the availability of programming language experts and/or tools.

Network effects lead to demand-sided economies of scale and to what is known as positive feedback respectively, increasing returns. Shapiro and Varian summarize this phenomenon as follows: “Positive feedback makes the strong get stronger and the weak get weaker” (Shapiro and Varian 1998, p. 175). Figure 2.1 shows this self-reinforcing cycle both for direct and for indirect network effects.

These network effects represent a huge motivation for users to select popular software products, and to prefer providers who can offer them strong network effects.

But there are other arguments in favor of large providers: first and foremost, they offer protection of investment, a key factor in light of the high switching costs

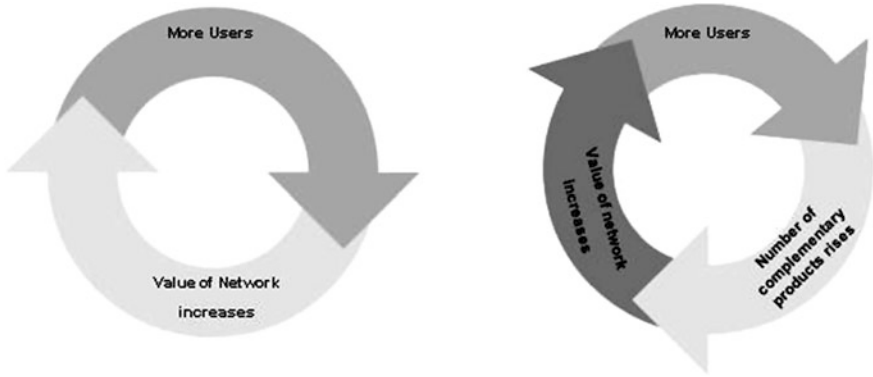


Fig. 2.1 Positive feedback loop—direct and indirect (modified from Bansler and Havn 2002, p. 819)

associated with software (see also Sect. 2.2.4). Risk reduction probably also plays a role: choosing software from a market leader such as SAP or Microsoft is unlikely to cost you your job. On the other hand, if an implementation project for open-source software ends in failure, those responsible will find it much harder to justify the decision, and failure is more likely to affect their standing within the company.

It is necessary to differentiate between the network utility and the stand-alone utility that software can deliver, i.e., the utility of the software irrespective of how many other people use it. An example of a good with both network and stand-alone utility is a spreadsheet program. The stand-alone utility lies in the functionality provided, whereas the network effect utility derives from the ability to exchange files with other users (direct network effect) or to obtain advice on how to use the software (indirect network effect). An e-mail system, on the other hand, is an example of the software that offers only a network utility, because a single user on his or her own derives no utility from the system at all.

Figure 2.2 a user's utility function. The total utility is the sum of the stand-alone and network utility. In this diagram, a positive linear correlation is assumed between the network utility and the number of users.

To measure the magnitude of the network utility in comparison to the stand-alone utility, we introduce a network effect factor Q . In addition, c represents the network utility and b the stand-alone utility of a software product. We now define the network effect factor as follows:

$$Q = \frac{c}{c + b}$$

where Q is standardized between zero and 1. The higher the value of Q , the greater the impact of network effects in comparison to the stand-alone utility. For example, the network effect factor of standard software such as Microsoft Excel

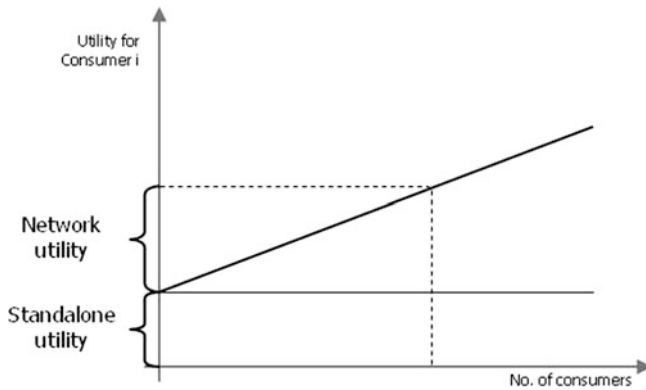


Fig. 2.2 Correlation between stand-alone and network utilities

would be somewhere between zero and 1. However, an EDI standard does not offer any stand-alone utility, so the network effect factor in this case is 1. We will return to this network effect factor in the context of assessing pricing strategies (see [Sect. 3.3](#)).

We will now examine the impact of network effects on software markets.

2.2.2 Impact of Network Effects on Software Markets

Let us begin with a straightforward example: Imagine that a newly established company has succeeded in developing a software product that offers superior functionality to Microsoft's Office package. In other words, the product offers the customer a higher stand-alone utility. Would this start-up be successful in penetrating the market? We would not like to bet on it. The problem for the small company is that the market leader—in this case, Microsoft—already offers its customers the benefit of high network effects. From a macroeconomic point of view, the optimum solution—assuming that switching costs are not too high—would be for all users to opt for the new software. This is because, as explained, if it offers a greater stand-alone utility, and if all users were to switch over; corresponding network effects would arise over time. Nevertheless, a wholesale changeover is unlikely to happen, because the existence of network effects can lead to a lock-into, a technically inferior technology (cf. David 1985; Liebowitz and Margolis 1994). As a result, a company that has an installed base in a network market has such a significant competitive lead that is extremely difficult for other players to catch up. In our example, Microsoft has a competitive lead.

2.2.2.1 The Penguin Effect

According to Farrell and Saloner, the fact that the installed base often hinders the changeover to a technically superior standard is due to informational and resulting

coordination problems (Farrell and Saloner 1985). Their reasoning goes like this: The sum of the utilities enjoyed by all market players could be increased, if these opt to transition to a new (technically superior) standard—in our example, if they choose the startup’s software. However, the users are unsure whether this transition is actually taking place. A potential switcher with incomplete information is facing the problem that the other market players may not follow suit, and the greater stand-alone utility enjoyed when using the new standard cannot compensate for the network utilities forfeited. The uncertainty about how the other market players will respond can encourage a company to maintain the status quo. This coordination problem is also termed the penguin effect, based on the following analogy: hungry penguins are standing at the edge of an ice floe. Out of fear of predatory fish, they hope that other penguins will jump into the water first, to check out the risk of falling victim to a predator. As soon as some of the birds have taken the plunge, the danger for the others is reduced, and the free-rider penguins follow suit (Farrell and Saloner 1987).

In light of this effect, software providers face a startup problem. Although every new player in a market has difficulties to contend with, they are more significant for companies in markets with network effects. The startup company in this scenario not only has to effectively promote the product itself, but also assure potential buyers that it is going to prevail in the marketplace and generate network effects. This startup problem caused by the penguin effect is also referred to as “excess inertia”. The converse problem, i.e., “excess momentum”, or an excess of different standards, can also arise.

The penguin effect is one of the main reasons why an inferior standard often manages to gain the upper hand; there are many examples of this phenomenon:

For a long time, the VHS video standard dominated the market, although Betamax was technically superior.

OSI protocols did not succeed in becoming standard at all levels, despite their high quality in technical terms. Today, Internet protocols dominate the market.

The QWERTY keyboard layout is said to be less efficient than alternatives, such as the later Dvorak layout, which conforms with ergonomic principles. Converting to a potentially more efficient layout is prevented by both direct and indirect network effects.

What we learn from these examples is that it is very difficult to replace a standard, once it has become firmly established on the market.

2.2.2.2 Diffusion Processes: The Model Proposed by Arthur

According to Arthur, small chance events can determine which standard ultimately prevails (Arthur 1989). The term “path dependence” describes the effects of earlier, sometimes random events that occurred during the diffusion process on the market structure (David 1985, p. 322). The model described below clearly illustrates how competition between two technologies plays out when both of them generate network effects.

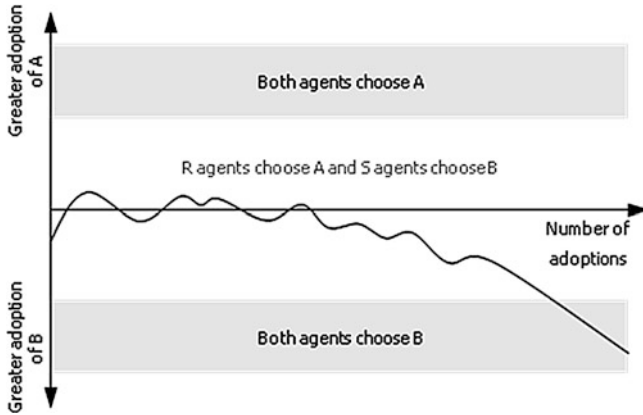


Fig. 2.3 Path dependence due to increasing returns (Arthur 1989, p. 120)

The model takes two technologies, A and B, as a starting point. In addition, there are two types of users, which Arthur terms R and S agents. The total utility of a technology is then the sum of the stand-alone and network utilities. The model uses the following parameters to denote the stand-alone utility:

- a_r stand-alone utility of technology A for R agents
- a_s stand-alone utility of technology A for S agents
- b_r stand-alone utility of technology B for R agents
- b_s stand-alone utility of technology B for S agents

We shall assume the following: technology A has a higher stand-alone utility for R agents than technology B ($a_r > b_r$). Conversely, technology B has a greater stand-alone utility for S agents than technology A ($a_s < b_s$).

As shown in Fig. 2.2, we assume that network utility increases linear with the number of users. n_a denotes the number of users of technology A and n_b the number of users of technology B.

Arthur's model shows how, based on the agents' decisions, a technology comes to dominate the market. To this end, Arthur develops a simulation model in which an R or an S agent opts for one of the two technologies at random. The agents choose the technology that offers them the greatest total utility. The total utility of technology A for the R agent is $(a_r + r \dots n_a)$, while the corresponding total utility of technology B is $(b_r + r \dots n_b)$. The parameter r stands for the marginal utility generated by each additional user. It is assumed that the S agents take their decisions on the same basis. Figure 2.3 shows the results of applying this simulation model.

In the white area shown in the diagram, R agents choose technology A and S agents opt for technology B, i.e., each chooses the technology that gives them the greatest stand-alone utility. However, if many S agents choose technology B, the network utility of technology B increases. From a given number of users upward, this prompts R agents to opt for technology B, too, as the higher network utility of

this technology more than compensates for the lower stand-alone utility for the R agents.

This simple model shows two things clearly: first, so-called early adopters play a decisive role in the struggle for market share. Second, that it is often chance events that determine how a given technology fares in markets with network effects.

2.2.3 Structure of Software Markets

For a long time, economists have recognized that network effects often lead to monopolies. For this reason, markets displaying strong positive network effects and feedback loops are frequently termed “tippy”: “When two or more firms compete for a market where there is strong positive feedback, only one may emerge as a winner. It’s unlikely that all will survive.” (Shapiro and Varian 1998, p. 176). Multiple standards or technologies seldom exist side by side. A dominant technology crowds out the others (Bessen and Farrell 1994, p. 118). Therefore, they are known as winner-takes-all markets.

An observation of today’s software markets bears out these theoretical considerations. Looking at markets for standard software solutions, for example, it is obvious that the number of providers has fallen sharply. A few years ago, the market still had room for alternatives to Microsoft products for browsers or office solutions, such as Netscape Navigator, the WordPerfect word processing system, or the 1-2-3 spreadsheet application. Now, there is really only one powerful competitor to Microsoft on these markets: the open source community.

The browser wars

The browser war between Microsoft and Netscape started in 1995. Netscape rose to prominence with its Netscape Navigator web browser. Prior to 1995, it had a market share of more than 80 %. The battle between these two companies commenced when Microsoft finally recognized the growing importance of the Internet, and developed its competing product, internet explorer (IE). During these wars, there were two issues of particular interest that will be described in more detail below: first, the importance of open-source products to rival commercial software, and second, the effect of price bundling on a business.

Phase I: The beginnings of the WWW

One of the leading web browsers to emerge in the early 1990s, when the Internet was really taking off, was Mosaic. It was the only browser to boast a graphical user interface that enabled users to surf the web. Established in late 1993, Netscape launched the first version of its Netscape Navigator browser in 1994. By 1995, Netscape had built up a monopoly on the browser market.

Phase II: 1995–1998

When Microsoft decided to enter the Internet market, the first version of the Internet Explorer was developed. The company had two decisive advantages over Netscape: first, Microsoft had far more financial resources at its disposal to develop browser software. Second, Microsoft was able to create product packages, and started to include Internet Explorer in its software offerings. From Windows 98, Internet Explorer was a fixed component of the operating system (along the lines of: “If it’s installed, it will be used”). Since Windows is preinstalled on about 95 % of all new PCs, Microsoft’s price bundling strategy soon paid off. In the following period, Internet Explorer’s market share rose from below 3 % initially, to over 95 %. In 1998, with its market share below 4 %, Netscape was finally compelled to admit defeat. Netscape then published the source code of its Navigator and converted it to the Mozilla open-source project.

Phase III: 2004 to the present

This phase is also referred to as the second round of the browser wars. In 2004, more and more security gaps in Internet Explorer were coming to light enabling Mozilla to regain market share. The Mozilla Organization launched Firefox 1.0 (a slimmed-down browser) in November 2004. It spread rapidly thereafter, not only because of Internet Explorer’s security deficiencies. Firefox offers a range of practical functions (tabbed browsing, find-as-you-type, Download Manager, etc.), which were contributed by open-source developers. Microsoft failed to upgrade the outdated technology underlying Internet Explorer 6.0 fast enough to prevent the modern, open-source browser, Firefox, from capturing significant market share. It was not until version 7 (many of whose “new” functions were copied from Firefox) that Internet Explorer regained a positive image on the market.

A monopolistic market structure such as that for office software often poses problems for users. However, Microsoft did not come under criticism for imposing a (higher) so-called Cournot price—as predicted by traditional microeconomics. Rather, it was for violating competition law that the European Commission levied a large number of fines on Microsoft, and instructed the company to disclose interface specifications and debundle certain of its products or face further penalties. Both demands benefit the other software providers by making it easier for them to develop and market products that are compatible with Microsoft’s standards.

2.2.4 Network Effects as a Competitive Factor

Our discussions so far indicate that network effects can represent a decisive competitive advantage for a provider. Once a software solution has become widely

adopted, a lock-in effect will occur. As a result, potential rivals are faced with a start-up problem whose magnitude correlates with the network effect factor.

At the same time, an analysis of the competitive environment must take into account that a user who changes to a different software solution generally incurs high switching costs. This applies particularly in relation to ERP systems, which explains why they are seldom replaced in practice. But what accounts for these high switching costs? One of the principles behind the ERP market is that this type of software models and shapes the user's business processes. Changing to a different provider would therefore result in considerable costs in terms of organizational changes. In addition, especially for standard software with a high network effect factor, there is uncertainty about how many other users will change over to a new software standard—compare the penguin effect discussed in [Sect. 2.2.2.1](#). The switching costs will also increase, the longer an ERP software product has been in use, because over time, it will become better integrated into the overall environment. Extensive system customizations increase the switching costs still further.

How can a software provider motivate users to switch over despite the lock-in effect? One possibility is to subsidize the change. Moreover, the vendors could pursue a low-price-strategy (see [Sect. 3.3](#)).

Major providers who are already able to offer their customers network effects may find it lucrative to maintain their competitive edge by not disclosing interface specifications in full. This makes it difficult for other providers to offer compatible products. In today's world, however, putting limits on compatibility is unlikely to be popular with customers. Users have long since realized that open standards make them more vendor-independent and above all, can simplify IT integration both within and between companies.

However, the product compatibility question pans out very differently for small software companies and for providers whose solutions have yet to be launched. These providers have no choice but to either go for completely open standards or develop products that are compatible with those of the major players. A very good example is provided by Business Objects, a provider of business intelligence software:

Business Objects was established in 1990 by two Oracle employees. The company began with an application that enabled Oracle customers to create database queries intuitively, without any knowledge of SQL. In the early years, the company targeted only Oracle customers and their application only supported Oracle databases. In other words, they pursued a niche strategy and did not include other databases at first. By concentrating on Oracle customers, Business Objects was able to forge a close partnership with Oracle. Oracle saw Business Objects products as complementary to its own; they were not rival offerings, but helped drive sales of its own database technology. Later, Business Objects built other partnerships, such as with IBM, and became a well-known provider of business intelligence technology. In 2007, the company was acquired by SAP.

The Oracle database system first served as a springboard for the products and services offered by Business Objects. This model can be found in many other areas of the software industry, such as in the games sector, which we will look at below.

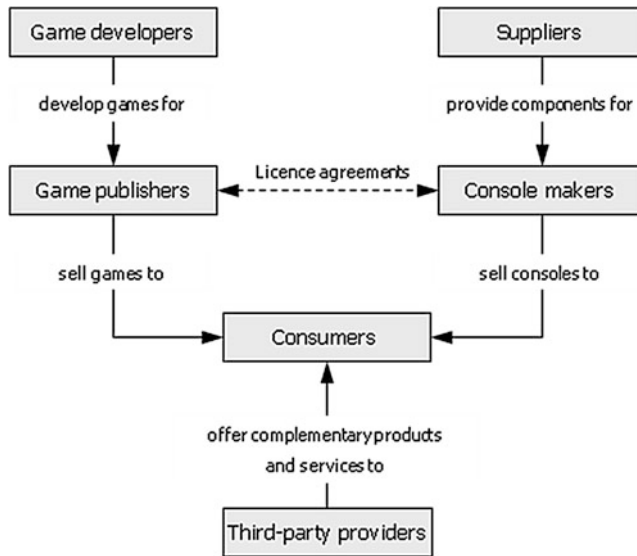


Fig. 2.4 Structure of the value chain in the digital games industry

2.2.5 A Case Study: Two-Sided Network Effects and Platform Strategies in the Digital Games Industry

In this section, we will describe a branch of industry positioned somewhere between the software and the media industry: digital games. We will use this market to explain the impact of two-sided network effects and the importance of platform strategies. Our focus will be on console games, because the varied results of network effects are particularly apparent in this market.

2.2.5.1 Overview of the Digital Games Industry

Digital games have grown into a sizeable market for the entertainment industry. The USA plays a leading role, not only due to the size of its retail market, but also because most of this sector's software providers are based there.

We will start by looking at the players and the value chain in the digital games industry. The main players are console manufacturers, suppliers, game developers, game publishers, consumers, and third-party providers. The value chain in this sector is shown in Fig. 2.4.

Console manufacturers develop and sell the consoles needed to play the digital games. For this reason, we refer to this hardware below as platforms. At present, the main console vendors are Microsoft, Nintendo, and Sony. They are assisted by *suppliers* who provide certain console components. For example, specialized sound and graphics chips and CPUs are produced by manufacturers like NVIDIA, ATI, IBM, and Intel.

Game publishers, such as Electronic Arts and Activision, reproduce and market games for these consoles. They act as intermediaries between the game developers and the console makers.

The games are developed by in-house or independent *game developers* or by studios such as Pyro Studios, Rockstar North, and Deck13. Either the developers are commissioned and funded by the game publisher to create a particular game, or they themselves approach publishers with a prototype. A unique feature of this market is that game publishers require a license from the console maker before they are allowed to market a game for that manufacturer's console. Under this model, the console manufacturer receives a portion of the publisher's sales revenue and thus has a stake in a game's success.

Third-party providers (e.g. manufacturers of peripheral equipment, magazine publishers, or video rental stores) offer complementary products and services such as gaming magazines, or joysticks, and memory cards. *Consumers* purchase consoles, games, and other complementary products and services via corresponding distribution channels (department stores or online retailers).

2.2.5.2 Two-sided Network Effects on Digital Games Markets

As mentioned above, the consoles represent the (industry) platform on which the games can be played. This constellation is comparable with other areas of the software industry. For example, operating systems underpin application software, while integration platforms such as SAP's NetWeaver and IBM's WebSphere form the basis for service-based software. As a rule, the platform providers do not have the resources to develop the applications, services, or games themselves (Cusumano 2004, p. 75). That is why they often work closely with other software and service providers. And very often, the winner-takes-all principle described earlier applies to platform/console providers, too. This has led to a consolidation on this market. In the early days, there were several console providers, whereas today there are only three: Microsoft, Nintendo, and Sony. The fierce competition on this market is often referred to as the "console wars". The manufacturers generally launch a new generation of hardware about every 5 years. Figure 2.5 shows the leading consoles on the European market. Some of them—such as Atari's Jaguar or Sega's Dreamcast—could not compete for very long, despite their technical superiority. Others—such as the Nintendo NES and the Atari 2600—held their own for years, despite the appearance of new, more advanced consoles. The network effect theory introduced earlier can explain why technically inferior systems can dominate markets.

The success of a console or platform does not depend on its technical merits or price alone, but also on the game software available for it. Therefore, this is a market with indirect network effects as the utility and popularity of a console is primarily determined by the availability of attractive games, which are complementary goods (compare Sect. 2.2.1).

The structure of the value chain in the digital games industry can also help us illustrate network effects of another kind, termed *two-sided network effects*.

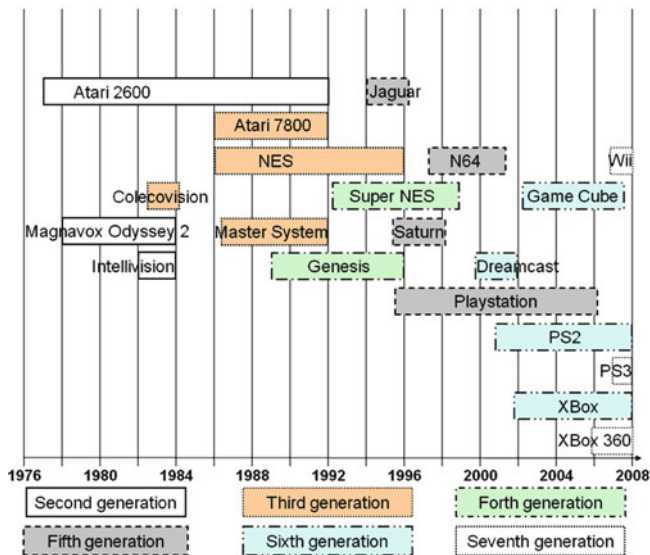


Fig. 2.5 Console generations in Europe (from Wikipedia, reworked)

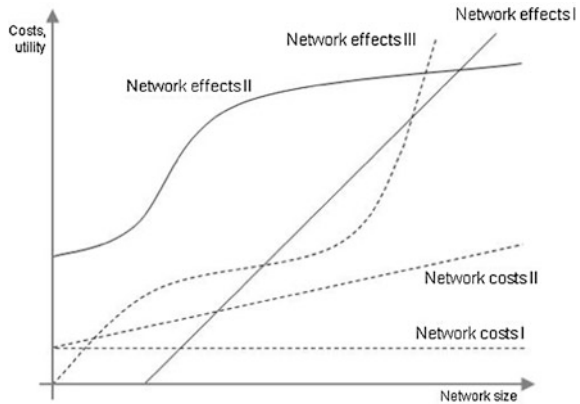
Multiple groups form around a given platform, whereby the utility that the platform offers one group depends on the number of members in the other group. Armstrong gives an illuminating example: “For instance, a heterosexual dating agency or nightclub can only do well if it succeeds in attracting business from both men and women” (Armstrong 2006, p. 1). On the digital games market, the two groups comprise game publishers and consumers. The game publishers benefit when large numbers of consumers own a console, resulting in high demand for the games played on it. On the other side of the coin, consumers benefit when many game publishers develop games for a console, because that increases the supply of games, which is the console’s indirect network utility.

In this section, we have restricted our discussion to the console games subsector, in order to illustrate how two-sided network effects work. There are, however, other sectors in the digital game industry, such as mobile games, browser games, and massive multiplayer online games, all of which are predicted to soar in popularity.

In recent years, we can observe the emergence of an increasing number of apps and games which are offered for social media platforms like Facebook. Market leader for the game industry in this sector is Zynga with titles like Farmville. Zynga is linked very closely to Facebook and generated approximately 12 % of Facebook’s revenue in 2011.

In the following section, we will describe the limitations of network effect theory, and possible ways of extending the theory.

Fig. 2.6 Possible cost and utility functions depending on network size (Weitzel 2004, p. 30)



2.2.6 Limitations of Network Effect Theory

Although network effect theory has given rise to some interesting insights with respect to explaining the widespread adoption of standards, there are limits to its applicability. In light of this fact, the following extensions, among others, have been suggested (Weitzel et al. 2000).

2.2.6.1 Homogeneous Network Effects and Costs of Network Size

An important potential extension of traditional network effect theory models relates to assumptions about the dependence of the network effect utility on the number of users. Often, as in the preceding sections, a linear utility function is assumed, i.e., every additional user leads to a constant utility increase for the users as a whole. However, degressive or progressive functions are equally conceivable. With the former, the n th user produces a smaller utility than the $(n-1)$ th user. With the latter type of function, the n th user produces a greater utility than the $(n-1)$ th. Similarly, various models also make differing assumptions about the cost functions in relation to the network size. Alternative functions are shown in Fig. 2.6.

However, all the utility functions shown in the figure—whatever form they take—ignore the individual nature of communications relationships. We believe that individualization is essential to decisions about employing standards, such as EDI. It is obviously extremely relevant to an automaker which standards its suppliers use. On the other hand, the company will be relatively unconcerned about the standards used by enterprises with which it has no business relationships at present and is unlikely to in future.

2.2.6.2 Type of Network Effect Utility

In the models proposed to date, the network effect utility is predominantly treated in the abstract and not specified. However, this means that the utility is not concretized, either. This raises the question as to what concrete utility, a given network effects actually offers.

2.2.6.3 Normative Implications

Most of the literature treats standardization decisions from a macroeconomic perspective, particularly with regard to the consequences of the existence of network effects in terms of market efficiency. The intention of most articles is to explain how users make standardization decisions. However, they do not, as a rule, offer any concrete recommendations.

The standardization problem touches on the limitations of network effect theory discussed above. First, a detailed, actor-specific method is proposed for modeling the costs and utility of network effect goods; second, the network effect utility is concretized; and third, recommendations will be offered to users on the extent to which standardization is the best solution.

2.3 The Standardization Problem

2.3.1 Approach and Background

The application systems used in many companies and groups have grown up over long periods, often in an uncoordinated manner. This gives rise to heterogeneous IT environments which, due to incompatibilities, hinder the flow of information between different parts of the organization. According to an oft-cited rule of thumb, developing and maintaining interfaces between incompatible subsystems can account for up to half of the entire IT budget. The use of standards is a key method of reducing these integration costs. Examples include the use of EDI standards to exchange information between companies, and SOA platforms deployed to enable the seamless integration of services from different providers on the basis of Web service standards.

An alternative to standards is the use of converters. However, this generally leads to higher costs than standards, because integrating n functions or business units can require up to $n * (n - 1)$ converters (Wüstner 2005). Furthermore, the consistent use of converter-based solutions usually reduces the flexibility of the overall IT environment. For these reasons, we will not discuss this option in any more detail.

The model introduced below looks into user decisions on the use of standards to link applications. However, providers can derive strategy recommendations from the model by employing it to anticipate the decisions, users are likely to make (Buxmann 2002). Like the model proposed by Arthur, we initially assume that a software standard offers a user both a stand-alone utility and a network effect utility. The stand-alone utility could be the functionality of an ERP system, for example. In principle, standard investment evaluation methods—such as multi-attribute utility analysis or the net present value method—can be used to compare the stand-alone utility of multiple software solutions. How can we concretize the network effect utility, which is largely treated in the abstract in network effect theory? In the following discussion, we will assume that using shared software

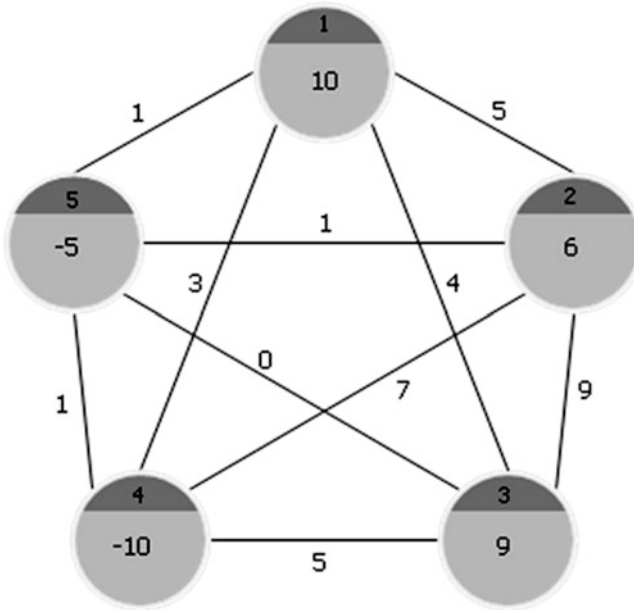


Fig. 2.7 Example illustrating the standardization problem

standards leads to savings in information costs. This term encompasses all costs incurred for exchanging information. For example, using a common EDI standard enables savings on human resources, postage, paper, etc. The common use of the same Microsoft Office solution generally saves money, time, and trouble when exchanging documents. Similarly, information cost savings can be achieved when a standard software product supports business processes across multiple departments.

On the other hand, costs are incurred for the procurement and implementation of software standards. These will be termed standardization costs in the following sections.

We will now simplify our reflections by taking a static approach. That means, for instance, that the cost of introducing a software standard will be assigned a specific value in our model. This is of course a simplification, because the costs associated with an investment of this kind differ from year to year. For example, users have to bear one-time licensing costs and also annual maintenance and service costs. This simplification is, nevertheless, unproblematic for two reasons: first, we can interpret the value assigned to the standardization costs as a discounted value of the payments incurred over the years; second, a dynamic approach that explicitly takes into account different points in time does not change the main model predictions that will be derived below.

The standardization problem is formulated on the basis of a graph as shown in Fig. 2.7. The actors are represented as nodes and the relationships between them as edges. The model is formulated in a general way: in other words, the nodes can stand for companies, e.g. in a supply chain, or organizational units within an enterprise.

The stand-alone utility and the standardization costs are modeled nodes related. The information costs that can be saved are edges related. If, to take the simplest case, we assume that there is only one standard available, meaning that each actor has to take a yes/no decision for or against introducing a given standard, we can label the stand-alone utility of node i as b_i and the standardization costs of node i as a_i ($i = 1, 2, \dots, n$). If the decision about introducing a software standard were to be taken with reference to the nodes and in isolation, only the difference between the stand-alone utility and the standardization costs would have to be determined for each node. If this net stand-alone utility is greater than zero, the software standard is a worthwhile investment, otherwise it is not.

But such an isolated treatment is not useful in practice, because the IT integration costs in particular are usually very high. In our model, the information costs along the edge from node i to node j are termed c_{ij} . These can be reduced if both actors implement the common standard, such as the same EDI standard, the same standard software or the same communications protocol. Of course, it is unrealistic to assume that the use of common standards will reduce the information costs to zero. But this does not pose a problem for the model, because the edge values may also be interpreted as the difference between the information costs without and with the common software standards.

Figure 2.7 shows a simple example. The respective net stand-alone utility is shown in the nodes, while the edges represent the communication costs that can be reduced. Using this example, we can demonstrate that, unlike classical network effect theory, this approach makes it possible to model the communications relationships between certain actors on an individual basis. That means the substantial demand for communications between nodes 2 and 3 is shown by edge costs of 9. These can be saved by adopting a common standard. In contrast, the need for communications between node 5 and the others is slight, and so using the common standard can only lead to minor savings in information costs.

In the first analysis, as shown above, the ideal case is when the standard is adopted by all of the nodes for which the net stand-alone utility is positive. In this case, this applies to nodes 1, 2, and 3. But standardization can be advantageous even for a node with a negative net stand-alone utility. This is precisely the case when this node has close communications relationships with others, and in consequence, substantial information cost savings can be achieved. In our example, this applies to node 4. Although the net stand-alone utility for this node is negative, standardization is worthwhile overall, as information costs amounting in 15 can be eliminated if nodes 1, 2, and 3 also introduce the standard. On the other hand, standardization makes little sense for node 5. Here, the information cost savings of 3 are lower than the negative net stand-alone utility of 5.

2.3.2 The Central Standardization Problem as An Optimization Problem

In formal terms, we can represent the standardization problem as an optimization calculation as follows:

$$\max F(x) = \sum_{i=1}^n (b_i - a_i)x_i - \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n c_{ij}(1 - x_i x_j)$$

s.t.

$$x_i \in \{0, 1\}$$

Decision variable x_i takes the value of 1, if actor i employs the standard. Only in this case can the stand-alone utility b_i be achieved, and only then will standardization costs a_i incur. The information costs c_{ij} can only be saved if both actor i and actor j employ the standard. The goal function also shows that the model tends toward optimization of the entire graph. In one application, for example, a central entity, e.g. the Chief Information Officer and his staff, could look for the optimal solution for the entire company. This, however, can mean that individual actors are worse off than before.

The above model only encompasses situations in which there is exactly one standard. In contrast, the extended standardization problem also looks at situations in which the decision maker can choose between several standards—we will not show the mathematical details of this model in this book (Domschke and Wagner 2005).

To illustrate the model, perform optimization and simulations, we have developed a prototype. Figure 2.8 shows the optimum standardization solution from a central viewpoint for a complete communications network with seven actors and a choice of three standards. In this example, the optimum solution is to implement multiple standards in the network, and even to provide individual actors with more than one standard.

The prototype makes it possible to generate communication networks with widely different structures and examine them under diverse cost distributions. For example, it is possible to investigate the effects of the network topology on the advantages of specific standardization options, such as partial standardization or full standardization. Figure 2.9 gives an example of this kind.

We have explored this model both by analytical means and via simulations (Buxmann and Schade 2007). This has permitted us to derive the following three basic propositions:

1. The best solution is often to standardize the network fully or not at all.
2. The larger the network, the more worthwhile full standardization will be.
3. If multiple software standards are available, it is seldom the best option to employ several of them.

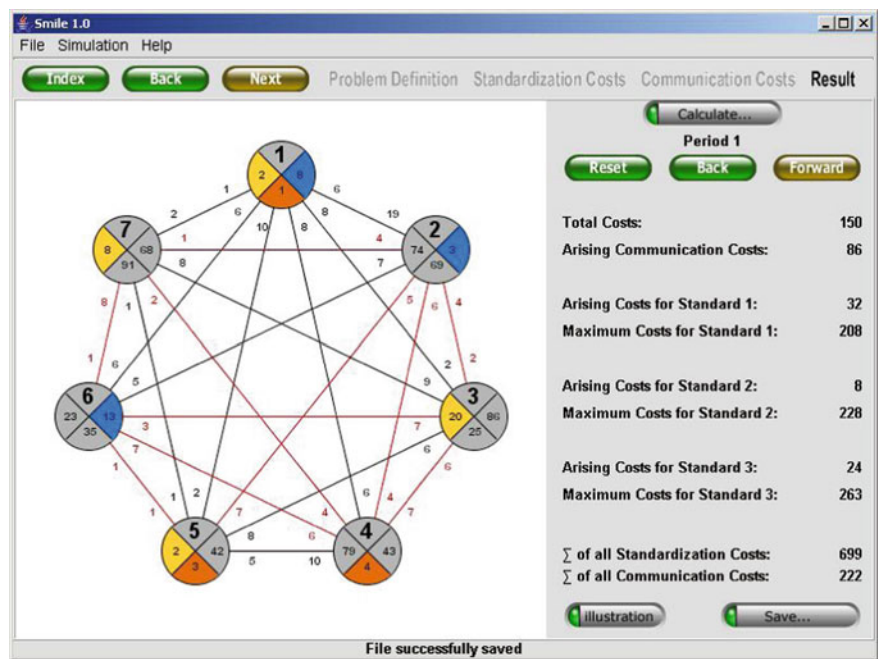


Fig. 2.8 Solution to the standardization problem for seven actors (Schade and Buxmann 2005)

That also means that best-of-breed solutions generally do not represent an ideal solution for the user organization.

This means that the best option for a major software provider is generally to offer solutions for all areas (nodes) of a network. A standard solution is usually beneficial to the users. This is especially true if the customer organization is large—expressed, in terms of the model, as the number of nodes n . One exception to this rule (which is, however, relatively rare in practice), is a scenario in which one area of the business (one node) has few, if any, information relationships with other areas (nodes). In this case, the customer’s best option may be to employ multiple software standards in accordance with a best-of-breed strategy.

Obviously, smaller software providers and niche providers must ensure their products comply with compatible standards and are therefore, attractive to users who wish to save on information costs (for integration or conversion).

2.3.3 The Decentralized Standardization Problem: A Game Theoretical Approach

Our considerations so far have been based on the assumption that a central entity exists that decides about adopting standards for all nodes. This would mean that a CIO and his team are able to choose (and enforce) the respective software standard

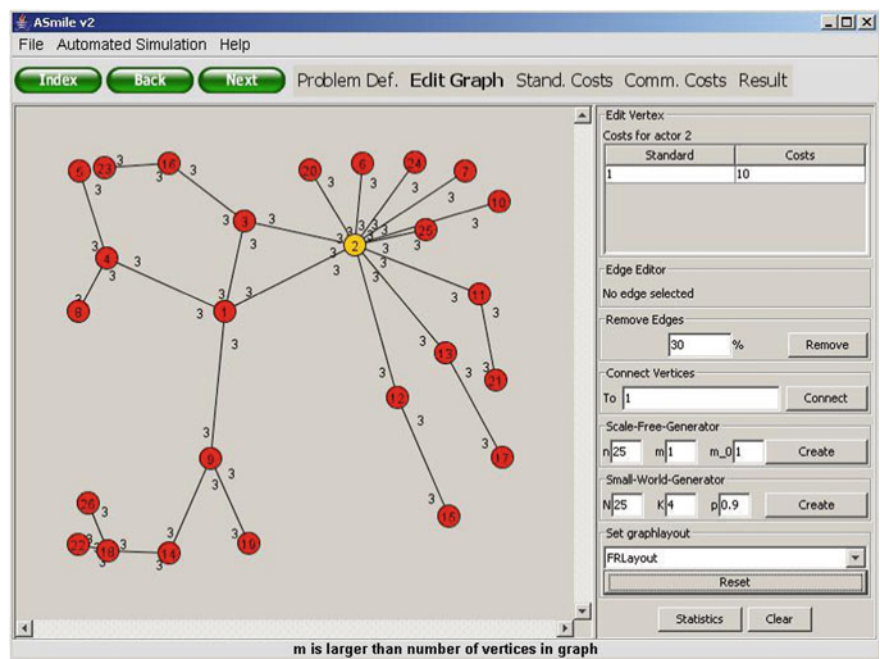


Fig. 2.9 Examination of various network topologies

for all organizational units. From an interorganizational perspective, it is reasonable to assume that the strongest player in a supply chain can impose the software standard on its partners. But what happens when every node takes standardization decisions independently (Buxmann et al. 1999; Weitzel 2004; Heinrich et al. 2006)?

To describe this decentralized problem, we will take an approach based on game theory (Weitzel 2004). We will simplify the problem, by assuming a single-standard problem and examining just two actors i and j . This is in fact perfectly sufficient to illustrate the differences between a centralized and a decentralized decision and the resulting consequences. Each of these actors takes a decision to implement or not to implement the given standard. We now introduce directed edges between the actors with the help of which we can show that adopting a software standard can benefit one participant more than another. In formal terms: if both actors adopt the given standard, actor i can save information costs c_{ij} , while actor j can save c_{ji} . This general problem can be modeled on the basis of non-cooperative game theory, as shown in Fig. 2.10:

This matrix may be interpreted as follows: the squares give the results of the four possible strategy combinations based on the two alternatives (standardize or do not standardize) facing actors i and j , respectively. Starting at the top left, if both actors adopt the software standard, both can save their information costs. What remains is the net stand-alone utility for actors i and j , which we will term u_i and u_j , respectively. In the top right-hand square, only actor i is adopting the

		Actor j	
		standardizes	does not standardize
Actor i	standardizes	u_i / u_j	$u_i - c_{ij} / -c_{ji}$
	does not standardize	$-c_{ij} / u_i - c_{ji}$	$-c_{ij} / -c_{ji}$

Fig. 2.10 The standardization problem in the light of game theory

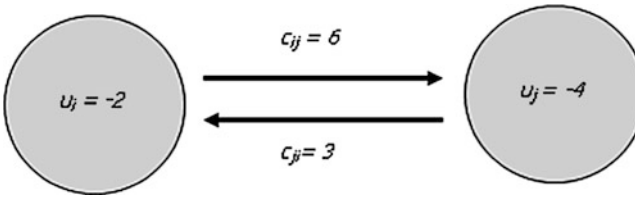


Fig. 2.11 An example scenario for decentralized standardization decisions

standard. As a result, actor i achieves $u_i - c_{ij}$ —the net stand-alone utility for introducing the software standard minus the information costs that actor i cannot save because actor j has decided against adopting the standard. In this scenario, actor j makes a loss to the amount of information costs c_{ji} . At the bottom left, the same applies, but the actors' roles are reversed. The bottom right square describes the case where neither of the actors adopts the standard. As a result, there are no standardization costs, and both actors have to bear the information costs c_{ij} and c_{ji} , respectively.

In game theory, a situation where no actor has an incentive to change his decision (in this case in favor of or against adopting the software standard) is known, in simple terms, as a Nash equilibrium. The Nash equilibrium is therefore a combination of strategies that represents a stable state. This game-theory-based matrix can now be examined with different parameter constellations to discover what equilibria form. For our purposes, a particularly interesting and instructive scenario is one where, from a global perspective, it would appear ideal for both actors to adopt the standard, but where, from a decentralized viewpoint, only one of the actors benefits from adopting the standard. Let us look at Fig. 2.11.

It is obvious at first glance that from a global perspective, the ideal situation would be for both actors to standardize. Although the net stand-alone utility is negative, i.e., the standardization costs are higher than the gross stand-alone utility, standardization is still worthwhile because information costs totaling 9 MUs (monetary units) can be saved. The negative net stand-alone utility, by contrast, only amounts to -6 MUs. So the total saving is 3 MUs. The problem now is that there is no central entity that takes the decision for both actors and seeks the optimum

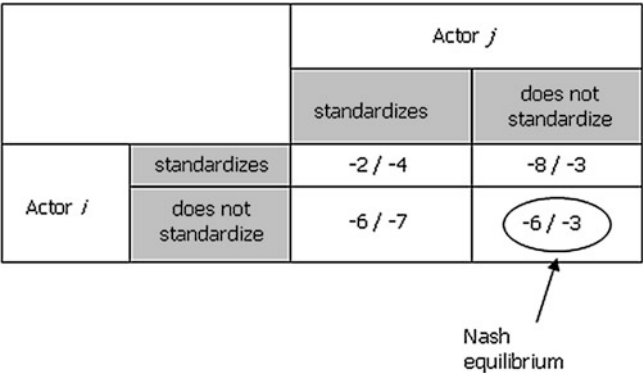


Fig. 2.12 Modeling the example using game theory

solution from a global viewpoint. However, assuming a decentralized perspective for both actors, it is clear that only actor *i* benefits from both sides adopting the software standard: the information cost savings of 6 MUs are greater than the negative net stand-alone utility. The picture is different for actor *j*, for whom adopting the standard does not make sense because *j*'s negative stand-alone utility is higher than the information costs that can be saved by adopting the standard.

Figure 2.12 shows the results matrix derived from game theory for this example. This matrix is interesting in that it shows that the standardization solution (top left square) cannot be seen as a Nash equilibrium, although this is the ideal solution from a global perspective. Obviously, actor *j* has an incentive not to choose this solution and save 1 MU.

2.3.4 The Standardization Problem: Lessons Learned

A comparison of the decentralized with the centralized model highlights the fact that decentralized decisions about adopting software standards tend to lead to a lower—and from a global perspective, frequently suboptimal—degree of standardization than centralized decisions. One reason for the actors' reluctance to adopt standards when faced with a decentralized standardization problem may be found in the penguin effect discussed in Sect. 2.2.2.1. A second is that different actors benefit to different extents when all of them adopt a standard.

In order to achieve an ideal result from a global perspective with decentralized decision structures, financial incentives could be worthwhile for those actors who lose out when the standard is implemented (Heinrich et al. 2006). For example, in the example given in Fig. 2.11, actor *i* could make a compensatory payment to actor *j* to encourage *j* to follow suit. This is problematic for two reasons: first, determining the appropriate size of the payment. Second, it will be all but impossible to negotiate compensation payments with all those many business partners and determine a distribution acceptable to all actors (see also Sect. 3.1.1 Co-operations

in the software industry). In practice, the stronger players tend more or less to dictate the standard to the weaker ones—it was during negotiations on EDI standards that the phrase “EDI or die” was coined. However, a recent empirical study showed that standards are often introduced by business partners cooperatively. The study found that in about 70 % of cases in the automotive industry, the major carmakers assisted their smaller partners, particularly systems suppliers, with the introduction of software solutions (Buxmann et al. 2005).

Furthermore, the standardization problem reveals that it is generally preferable for user organizations to implement just one standard. In consequence, internal network effects (e.g. within an enterprise or a supply chain) reinforce the position of major software providers. Attempting to introduce a large array of different systems, especially in large organizations, will often lead to high information costs, e.g., in the form of integration and conversion costs. In many cases, therefore, the best solution from a global perspective is to use an integrated solution from a single vendor—complemented, perhaps, with compatible products from smaller software providers and niche providers. Best-of-breed solutions are only advantageous for the user when there is very little exchange of information and so the information costs between different areas of the business are low—a rare scenario in the real world. A key issue for users and providers is to what extent service-oriented architectures and platforms can contribute to reducing communication costs through the use of open standards, and therefore make best-of-breed solutions more attractive (we will discuss service-oriented architectures in more detail in [Sect. 4.7.](#))

From a provider’s perspective, too, it is seldom worthwhile to incorporate a host of different standards into a solution (Plattner 2007, p. 3). It makes more sense to select the right standard in terms of giving customers interoperability between different vendors’ platforms. Against this background, it is, of course, vital for providers to participate in the development of standards in order to assert their strategic interests. However, during the development of standards, problems frequently arise for the following reasons (Plattner 2007, p. 3):

In standardization processes, providers often represent different economic interests (we have looked into this issue in [Sect. 2.2.4](#))

Standardization processes are often slowed down by the typical features of committee work (Plattner talks about “committee thinking”).

For employees, participation in standardization bodies is often merely an extra task in addition to their main jobs, which is reflected in the way they prioritize their activities.

2.4 Transaction Cost Theory: In Search of the Software Firm’s Boundaries

The position a company assumes within the value chain is of great importance. In essence, this is a classic make-or-buy decision: what should the company produce or develop in-house, and what should it source from third parties? Transaction cost

theory provides some interesting insights into these questions. In [Sect. 2.4.1](#), we introduce the starting point and elements of transaction cost theory. In [Sect. 2.4.2](#), we present a general model for economically efficient division of labor among companies. In addition, we demonstrate where transaction cost theory can be applied within software companies. [Section 2.4.3](#) deals with the impact on transaction costs of structural changes—such as those brought about by the emergence of new communications technologies. [Section 2.4.4](#) considers the way intermediaries can lower transaction costs.

2.4.1 Starting Point and Elements of Transaction Cost Theory

In markets characterized by division of labor, players are linked by many and varied exchange relationships. This is the starting point for transaction cost theory (Williamson 1985). But the theory focuses not on the exchange of goods and services per se, but on the transfer of property rights that precedes the exchange. This transfer is referred to as transaction. For example, a provider handing over a piece of tailor-made software to its customer is a transaction. Further examples include integrating a module developed by an offshore service provider, or forwarding functional specifications from department A to department B within a company. To avoid confusion, we would like to point out that the term “transaction” is also used by the software industry in the context of database systems. This, however, is completely unrelated to the way the term is used in economics.

Transactions cause costs—an insight, incidentally, that economists ignored for many years. In particular, costs are incurred while gathering information and communicating with the transaction partner during deal initiation, agreement, completion of the transaction, and for the purposes of control/monitoring and fine-tuning. Let us take a look at an example. To draw attention to its offering, a custom software vendor must participate in trade shows and conferences and make regular appearances in the trade press. The client will also want to gain an overview of the market. This means that both companies will incur significant costs associated with initiating the transaction (setup costs). Further costs will be incurred for negotiations as to the scope of service and the price of the software to be developed. Other costs include those for policing and enforcement (for example via testing) and potentially for later modifications (e.g. if requirements should change). All in all, the provision of custom software is a transaction cost-intensive process.

This is where transaction cost theory comes in. It is concerned with determining what organizational form incurs the lowest transaction costs in a given situation. The approach can be applied both to the division of labor between enterprises and among departments within a company. We will look at the division of labor among market players in greater detail below. But first, let us examine the major determinants of transaction costs, and briefly tease out the theory’s underlying assumptions.

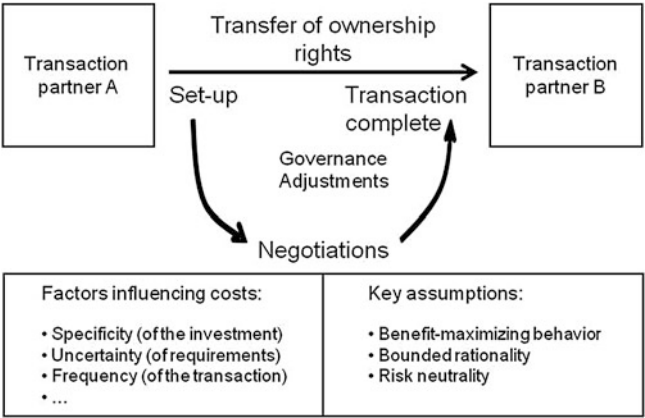


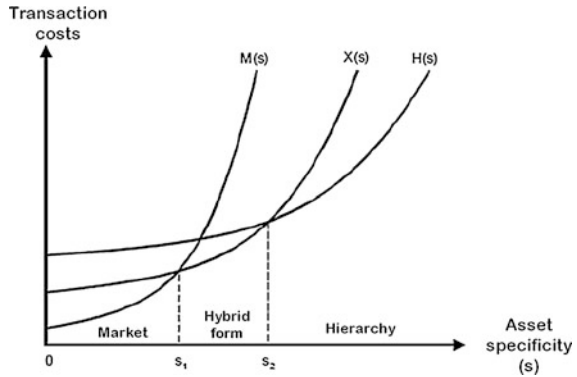
Fig. 2.13 Basic model underlying transaction cost theory

In transaction cost theory, the level of investment specific to the transaction (often termed simply “specificity”) is regarded as the main determinant of the cost of a transaction. The reasoning is quite straightforward. While the purpose of transaction-specific investments is to cut production costs, high transaction-specific investments create considerable dependency between the parties. Because once an investment has been made, the company cannot switch transaction partners without incurring losses, and it must be prepared to renegotiate terms with the existing partner. For example, large-scale outsourcing of IT services can create a dependency of this type. So clearly, transaction-specific investments will initially drive up transaction costs. The magnitude of transaction costs are also influenced by factors such as the degree of uncertainty associated with the transaction, the frequency, the strategic significance, and the atmosphere in which it is conducted.

Transaction cost theory assumes that all parties are characterized by opportunistic behavior, bounded rationality, and risk neutrality. These and other key assumptions are illustrated in Fig. 2.13.

Opportunistic parties can be assumed to act solely in accordance with their own interests. This means that if one of the partners has an opportunity to maneuver the other partner to renegotiate terms, he or she will usually do so. Certain information, such as compatibility with other solutions, is likely to be held back. What’s more, opportunistic players do not always fulfill their promises when other parties cannot easily determine compliance (e.g. regarding the quality of a software module). In the software industry in particular, judgments regarding a product’s quality can often only be made after the solution has been deployed—opening the door for opportunism. Bounded rationality in this context means that transaction partners do not have access to all available information (e.g. about a customer’s solvency).

Fig. 2.14 Transaction costs as a function of asset specificity (Williamson 1991, p. 284)



2.4.2 Division of Labor Among Companies: A Transaction Cost Theory Perspective

Probably the most well-known branch of transaction cost theory addresses the question of where an enterprise begins and ends. For simplicity's sake, a clear boundary is drawn between the company (also termed hierarchy in the model) and the market and (as a later addition)—cooperation as a hybrid of market and hierarchy. These three basic types of arrangement are characterized by the following coordination mechanisms: the price for the market, the organization for the company, and a mixture of the two for alliances.

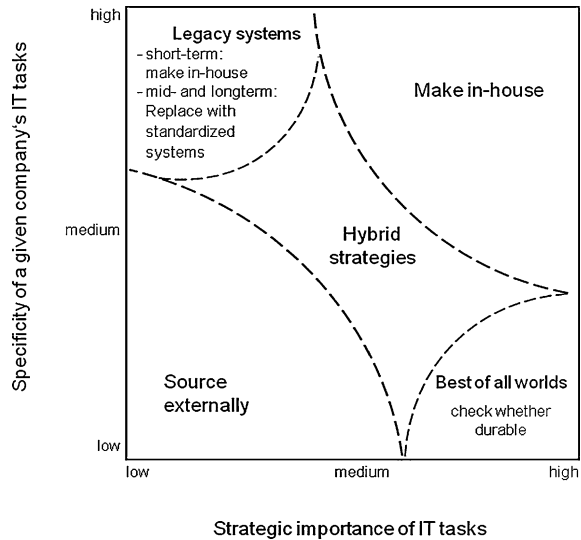
According to the theory, when transactions are not associated with transaction-specific investments (i.e. when they are unspecific), the market is the most efficient coordination mechanism. On the other hand, if the partners become dependent on one another through high transaction-specific investments, they have more incentive to act opportunistically, taking advantage of the other party's dependency. However, this opportunism can be effectively restricted by means of the hierarchy. This means that when specificity is high, hierarchy is the most efficient coordination instrument. Likewise, moderate specificity calls for cooperation as the coordination instrument, as it combines elements of markets and hierarchies.

Transaction cost theory has developed these ideas further, as illustrated in Fig. 2.14. In this representation, when specificity is below s_1 , the market is the most efficient coordination form. When specificity is between s_1 and s_2 , an alliance is the most efficient. And when specificity is greater than s_2 , the company is the coordination instrument of choice.

Figure 2.14 illustrates the transaction costs associated with various organizational forms, at different levels of specificity. However, the exact shape of the curve cannot be determined by analytical or empirical means. This is why it is discussed controversially in the literature.

Initially, transaction cost theory mainly focused on explaining the emergence of enterprises and alliances as alternatives to the market as regulating mechanisms—a macroeconomic question. Today, the theory has become a standard business

Fig. 2.15 Decision matrix for outsourcing IT tasks (Picot and Maier 1992, p. 21)



instrument for determining the boundaries of a given enterprise. It typically addresses make-or-buy decisions at the level of individual tasks, and relationships with the contractor. In the software industry, this topic and its many variations are of great importance—from outsourcing to specialists, to the use of offshoring. We will take a closer look at these questions in [Sect. 4.1](#).

Concerning software enterprises in particular, transaction theory provides valuable insights into the factors of greatest interest to the end customer, especially the business user. In contrast to make-or-buy decisions as mentioned above, this question has long been addressed exhaustively in the literature. For example, simple management instruments were developed that support decision-making concerning outsourcing IT tasks. Figure 2.15 illustrates the tool suggested by Picot and Maier.

In addition to specificity, Picot and Maier also consider the IT tasks' strategic importance—as we already mentioned briefly in the section describing the variables involved in transaction costs. In general, they recommend completely outsourcing only those IT tasks that have a relatively low specificity and a low strategic importance. Network operations fall into this category for many enterprises—which explains why so many companies' foray into outsourcing begins with this task.

2.4.3 Structural Changes to Transaction Costs: The Move to the Middle

We have already shown that transaction cost theory is helpful in the search for the most cost-effective form of organization for a given environment. In the real world, however—particularly in the software industry—the environment is by no means unchanging. New communications technologies and the rise of new nations with impressive IT skills and relatively low labor costs are bringing about radical

change. Against this background, we will sketch the impact of these changes on transaction costs and in consequence, on the selection of the most cost-effective form of organization.

Most market watchers today agree that new communications technologies such as the Internet and IP-based services are leading to a reduction in variable transaction costs and causing the curve in Fig. 2.14 to sink and flatten (Bauer and Stickel 1998). As a result, the point of intersection for the transition to the next form of coordination is moving to the right. This means that moving to cooperative or hierarchical relationships only begins to make sense in economic terms at a higher asset specificity. A pithy term for this trend is “move to the market”. The core proposition is that as a result of new communications technologies, coordination within an enterprise is giving way to cooperative relationships and markets, and that value chains are becoming more fragmented. For the software industry, this would mean that a large number of specialized businesses would arise—a scenario that could be reinforced by the new service-oriented paradigm (see Sect. 4.7).

However, although many companies are now collaborating more intensively than they did in the past, they are often doing so with few partners. This cooperation strategy demands transaction-specific investment, for example for agreeing and implementing special modes of data exchange. In accordance with transaction cost theory, this leads to an increase in asset specificity which would tend to counteract the “move to the market” (Clemons et al. 1993). In summary, both trends taken together mean that cooperation is beneficial, a phenomenon which is referred to as the “move to the middle”.

The new opportunities for outsourcing IT services to low-wage countries, which we discuss in Chap. 4, also play a key role. In terms of classical economic theory, this means that new providers are entering the arena, who will tend to reduce prices on the consumer market or make it attractive to outsource certain tasks that were performed more cheaply in-house until now. At this point, transaction costs must be taken into consideration. A software provider will only outsource services to a software company in India, for example, if the lower development costs resulting from massive wage differentials are not completely offset, and more, by higher transaction costs. Many factors can raise transaction costs, ranging from the greater effort required to develop software specifications and make modifications, to the additional costs associated with supervising the outsourcing provider. Even this brief list of potential issues makes clear that outsourcing software development tasks does not automatically reduce overall costs. Bad decisions can also be made because cost accounting methods typically only show up only a company’s production costs, but not the transaction costs. We will return to this issue in Sect. 4.1 of this book.

2.4.4 Excursion: Intermediaries and Transaction Costs

Transaction cost theory centers around transactions, which we defined earlier as the transfer of usage rights. Taking the transaction as the starting point of analysis, transaction cost theory offers insight into the form of organization that makes most

economic sense in terms of the size of transaction-specific investments. Aside from this, the concept of a transaction is used again and again in discussions of the economic importance and usefulness of intermediaries.

Until now, most intermediaries in the software industry have been marketing agents or distributors sandwiched between the developers and the users of a software product, particularly in business-to-consumer markets. In the near future, the trend toward splitting conventional standard software packages into modules and the emergence of service-based architectures will lead to intermediaries wielding unprecedented influence in other subsectors of the industry, too. Against this background, we will present the basic tenets of intermediation theory below, despite the fact that this is not directly connected to transaction cost theory. We will then return to intermediation theory in the sections on distribution strategy and industrialization.

Intermediaries are companies that do not make products or provide services themselves, but deliver a good or a bundle of goods to those with a demand for it. The most familiar example of intermediaries is retailers who offer consumers a large number of products from various manufacturers at a convenient location for their customers. But intermediaries can also be publishers or television companies that very often do not create content themselves but purchase it from freelance writers or specialized production companies and offer it in the form of a daily newspaper or a TV channel in accordance to customer preferences. We have already given various examples from the software industry. There is a place for intermediaries whenever conducting a transaction through them is more cost-effective than conducting one directly between the supplier and the customer.

Baligh and Richartz's mathematical model makes it possible to assess the effect of an intermediary, at least with regard to the search phase of a transaction (Baligh and Richartz 1967). Let us assume a market with m suppliers and n customers. If a customer wishes to gain an overview of what is on offer, it needs to ask all m suppliers. As this applies for each of the n customers, $m \times n$ contacts will be necessary. However, if an intermediary is involved, each supplier and each customer only needs to register its details with an intermediary, and only $m + n$ contacts are required. Even when n and m are relatively small, this means that employing an intermediary is preferable in terms of the number of contacts required, and this therefore makes good economic sense.

2.5 Software Development as a Principal-Agent Problem

2.5.1 Incentive-Compatible Compensation and Efficient Control

In the development of custom software as well as in implementation projects, maintenance work and the operation of standard software solutions, software companies and customers work closely together. This close collaboration gives

rise to complex dependencies which in economics are known as principal-agent problems. The rest of [Sect. 2.5](#) will present the underlying idea ([2.5.1](#)), show how it can be applied in practice through compensation schemes ([2.5.2](#)) and control systems ([2.5.3](#)).

2.5.2 Principal-Agent Relationships: Definitions and Basic Principles

The principle-agent theory (Spence and Zeckhauser 1971; Ross 1973; Jensen and Meckling 1976) focuses on the division of labor relationship between a principal and an agent. A relationship of this kind exists when, to realize its interests, a principle transfers powers of decision-making and execution to an agent and offers compensation in return.

For our purposes, the relationship between a software company (agent) and a company (principal) that commissions the former is particularly important, where the software company takes responsibility for developing, delivering, implementing or operating software and is paid to do so. A different principal-agent relationship is also of interest to us: if a software company outsources one part of the development process to another company, for example the development of a software module to a business located in a country where labor costs are lower, the software company is then the principal and the outsourcing company in the low-wage country assumes the role of agent (see also [Chap. 4](#)).

The principal-agent theory is based on two core assumptions. First, it is assumed that both the principal and the agent are motivated by utility maximization and systematically use discretionary powers in their own interest (e.g. by over-reporting the programming time taken). Second, it is assumed that the agent has an information advantage with respect to performing the task assigned. The agent generally knows better than the principal, for example, how many person-months a development project will require, and will often not reveal its true interests (e.g. to lock the principal into a long-term relationship). This means that the principal and the agent act rationally in terms of the (incomplete) information available to them, which is termed bounded rationality. [Figure 2.16](#) gives an overview of this basic model.

Opportunistic behavior and bounded rationality in the principal-agent relationship can result in three types of problems which also describe the relationship's structure in temporal terms.

Hidden characteristics arise before conclusion of the contract. Initially, the principal is not fully cognizant of the quality attributes of the agent and the services he or she delivers. This situation involves the risk that the principal could select an unsuitable agent. That is the case when the customer commissions a software company that does not possess the knowledge and skills needed to develop a particular solution.

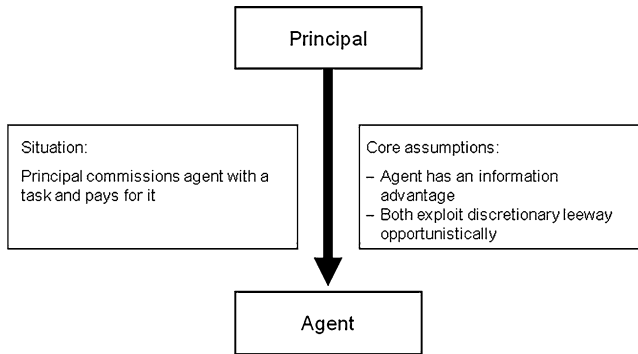


Fig. 2.16 Principal-agent relationship

Hidden action describes the risk that the principal cannot observe or cannot evaluate the agent's actions. In both cases, the principal knows the end-result but has no way of assessing to what extent this is due to the agent's efforts or external factors. As a result, the agent can always exploit his or her freedom to its own ends. For example, a customer will find it very difficult to discover what quality assurance tests have been performed and what the outcome was.

Hidden intention denotes the danger that the principal can become dependent on the agent because it relies on the services the agent performs. Like hidden action, hidden intention only arises after the contract has been signed. The agent can exploit this fact to its own advantage. For example, if a customer concludes an outsourcing agreement, it will automatically be dependent on the outsourcing provider.

Hidden characteristics, hidden action and hidden intention lead to so-called agency costs for the principal. To reduce these costs, the principal can use mechanisms such as compensation plans and control systems. Both these approaches will be described below, and we will provide insight into both quantitative and qualitative research on principal-agent theory.

2.5.3 Incentive-Compatible Compensation Schemes

Performance-based contracts or contracts for services are the basis for compensation of services rendered. The form a compensation scheme takes depends on the type of contract involved. Compensation schemes for both types of contract are described below (Picot and Ertsey 2004).

2.5.3.1 Compensation Based on a Contract with a Defined Result

By signing a performance-based contract, the service provider undertakes to deliver a defined result with clearly specified characteristics in accordance with

agreed terms and conditions. So only the price remains to be set. There are two ways of doing this: The customer can either recompense the provider for the actual costs incurred plus some additional amount (“cost-plus”) or pay a fixed price. With cost-plus contracts, the customer alone bears the project’s cost risk. The provider is in a position to report inflated costs, and even clauses stipulating the disclosure of cost information cannot rule this out completely. A cost-plus contract makes sense when the project specifications are incomplete and the provider is not able to estimate the production costs.

However, if one of these two conditions is not fulfilled, a fixed price agreement is advisable. Under an agreement of this kind, the customer pays a previously agreed fixed price at the end of the project. This means that the provider alone bears the projects’ cost risk. If the costs incurred by the provider are below the agreed price at the end of the project, the provider has realized an (additional) profit. But if the total project costs exceed the agreed price, its profit margin shrinks correspondingly and in the worst case, it might even make a loss. A fixed price certainly gives the provider an incentive to conduct the project efficiently. At the same time, however, there is no motivation to pass on any savings to the customer. We will discuss methods of estimating costs and approaches to pricing custom software projects in [Sect. 3.3.4](#).

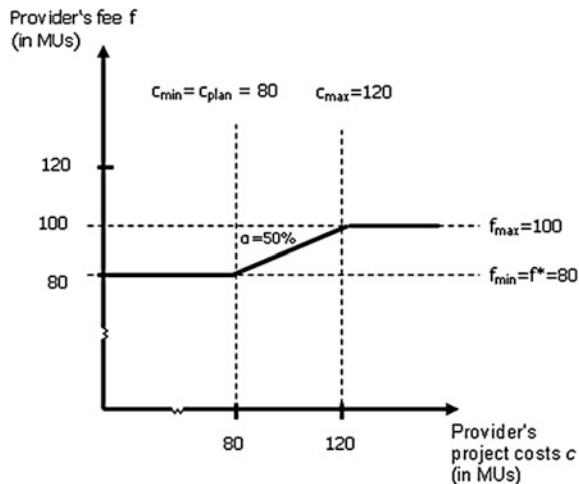
Both types of contract feature a completely one-sided risk situation, with only one of the two parties bearing all of the risk. This gives the other side leeway to act opportunistically. The model presented below describes the distribution of risk for cost-plus contracts. Let us assume that a provider receives compensation e . This comprises a fixed fee f_{\min} and a certain share α of the difference between planned and actual costs $c_a - c_p$, where c_p represents the planned costs and c_a the actual costs. The customer and the provider therefore share the excess costs. If c_a is below the contractually defined minimum c_{\min} , then the provider still receives the minimum amount f_{\min} —thereby realizing additional profit. But if c_a exceeds c_{\max} , the provider still receives the agreed maximum amount f_{\max} —and its profit margin sinks, or it may even make a loss. The compensation scheme, which limits risk for both parties, is as follows:

$$f = \begin{cases} f_{\min} & \text{where } c_a < c_{\min} \\ f_{\min} + \alpha \cdot (c_a - c_p) & \text{where } c_{\min} \leq c_a \leq c_{\max} \\ f_{\max} & \text{where } c_a > c_{\max} \end{cases}$$

Figure 2.17 illustrates this compensation scheme with an example in which c_p and c_{\min} both amount to 80 monetary units (MUs).

Additional development costs up to a maximum of 120 MUs are contributed by both parties equally, i.e., for $c_a = 100$ MUs, the provider receives 80 MUs plus $0.5 \cdot (100 \text{ MUs} - 80 \text{ MUs}) = 90$ MUs. Project costs in excess of 120 MUs are borne solely by the provider. If the project costs are below 80 MUs, the provider alone profits from the efficiencies achieved.

Fig. 2.17 Example compensation scheme based on a contract with a defined result



2.5.3.2 Compensation Based on a Contract for Services

Under a contract for services, the provider undertakes to perform a particular task such as programming or the provision of a service. If a fixed price is agreed, the provider has an incentive to work cost-effectively and may not focus sufficiently on the quality aspect—after all, it has only agreed to perform a task. This problem can be addressed by means of an incentive-compatible compensation scheme in which the fee is linked to the quality of work performed.

Let us refer to the provider's performance (e.g. number of lines of code) as p_a . If p_a is below an agreed threshold p_{\min} , the provider does not receive any fee—as a deterrent. In the interval $[p_{\min}; p_{\max}]$, the service provider receives the minimum fee f_{\min} and a share of the bonus b which is equal to $f_{\max} - f_{\min}$. If the performance is greater than p_{\max} , the provider only receives the maximum fee f_{\max} , so no provider will be motivated to achieve that. If the bonus paid in the interval $[p_{\min}; p_{\max}]$ is to increase linearly with the performance, the overall fee f is calculated as follows:

$$f = \begin{cases} 0 & \text{where } p_a < p_{\min} \\ f_{\min} + \frac{p_a - p_{\min}}{p_{\max} - p_{\min}} (f_{\max} - f_{\min}) & \text{where } p_{\min} \leq p_a \leq p_{\max} \\ f_{\max} & \text{where } p_a > p_{\max} \end{cases}$$

In practice, this depends on the ability to measure performance p_a in an objective way. For a software development project, this can be based on lines of code while for network services, service level agreements can be employed.

We will illustrate this scheme by way of a brief example. Let us assume that the performance of a software company is to be measured in lines of code (LoC). Although this indicator is not ideal for various reasons, it can still give a rough idea of the provider's performance and is used, for example, in models designed to estimate software development effort, such as COCOMO.

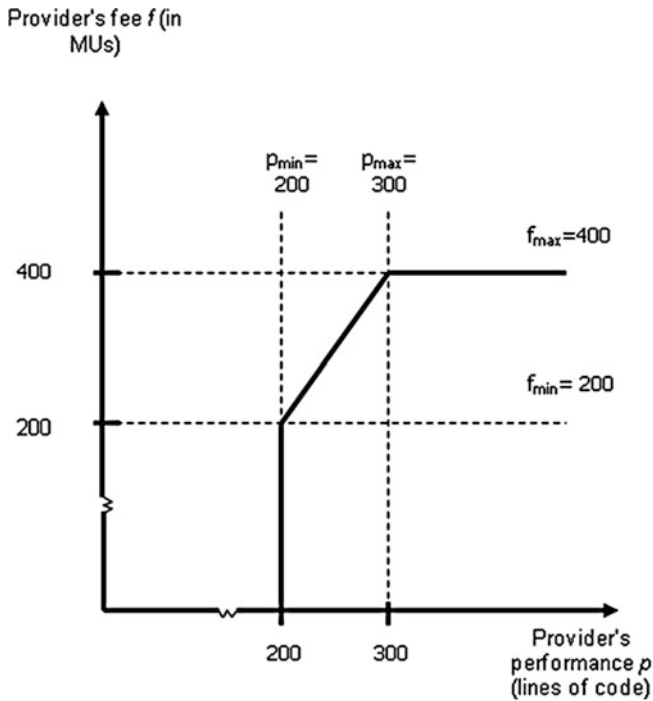


Fig. 2.18 Example compensation scheme on the basis of a contract for services

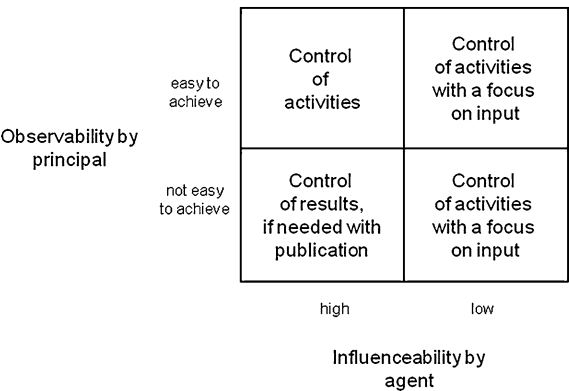
In our example, we assume that the contracted software company is supposed to develop at least 200 LoC and receive a fee of 200 MUs for 200 LoC. Up to 300 LoC, the fee increases linearly up to 400 MUs. Below 200 LoC, the software company receives no fee at all. Above 300 LoC, it always receives 400 MUs. The customer is therefore introducing two separate incentives: at least 200 LoC are required (e.g. to avoid jeopardizing the follow-on project) and up to 300 LoC would be very useful (reflected by the fee for each additional LoC in the interval $[200;300]$ MUs. Figure 2.18 gives an overview of this example.

To conclude, we would like once again to highlight the basic differences between the two compensation schemes presented. Both include the idea of sharing risk within given limits. But the crucial difference is the basis on which the fee is determined: for a contract with a defined outcome, this is the costs incurred by the provider, whereas for a contract for services, it is the provider's performance.

2.5.4 Control Systems

Another way to reduce agency costs besides compensation schemes is to deploy control systems. The design of a control system depends on whether the principal controls only the results produced by the agent or how the agent goes about its work or the input factors such as the development methodology employed.

Fig. 2.19 Various control system designs (based on Hess and Schumann 1999, p. 360)



Research findings on the efficient design of control systems show that the main factors determining the selection of a suitable control system are the agent’s ability to exercise influence and how well the principal can observe what is happening (Ouchi 1977; Picot 1989). Figure 2.19 shows the underlying logic.

Figure 2.19 provides a basis for a general recommendation for software development. Normally, the provider has many ways of controlling the involvement of employees in the project. For example, a custom software provider can choose different project leaders and programmers for a project. It is reasonable to expect that the quality of results will depend on the experience and training of these project members. Given this fact, the agent is able to exert a large influence, so that the left-hand column of Fig. 2.19 applies. However, it is normally difficult for the principal to observe and evaluate the agent’s efforts. Even if the customer understands the software development process per se, perhaps because he himself is a member of an IT department, he seldom has truly reliable information about the concrete status of the project. So in the final analysis, he has no alternative but to limit himself to controlling the results and possibly threatening to acquaint the industry at large with the provider’s failures—this is why reputation is sometimes referred to as the customer’s collateral.

The methods of limiting agency costs discussed above are based on a series of assumptions that simplify the real state of affairs. For example, when formulating the principal-agent problem, whether with the software company as the provider or the customer in the case of sub-contracts, we assumed that the specifications are created once and then used by the provider as a basis for developing the software. A far more realistic assumption is that the customer needs to be involved repeatedly and at different points in the development process. A further potential complication is that the people who conclude the contract and those with the required knowledge and skills may come from different parts of the customer’s organization, typically the IT department and the department that will actually be using the software, which are unlikely to have the same interests. These two influencing factors lead to much more complex principal-agent relationships than those presented here.

The Software Industry
Economic Principles, Strategies, Perspectives
Buxmann, P.; Diefenbach; Hess, Th.
2013, XII, 224 p., Hardcover
ISBN: 978-3-642-31509-1