Middleware Platform Pricing

Mat-2.4108 Independent research projects in applied mathematics

Jenni Brunila
48376C
## Contents

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents .................................................................</td>
</tr>
<tr>
<td>1 Introduction ...........................................................</td>
</tr>
<tr>
<td>2 Software pricing methods ...........................................</td>
</tr>
<tr>
<td>2.1 Value of the software .............................................</td>
</tr>
<tr>
<td>2.2 Licensing ...............................................................</td>
</tr>
<tr>
<td>2.3 Used licensing methods ............................................</td>
</tr>
<tr>
<td>2.3.1 Packaged perpetual licenses ....................................</td>
</tr>
<tr>
<td>2.3.2 Server based licenses ...........................................</td>
</tr>
<tr>
<td>2.3.3 Network licenses ...................................................</td>
</tr>
<tr>
<td>2.3.4 Unbundled licensing ...............................................</td>
</tr>
<tr>
<td>2.3.5 Risk-sharing licensing ...........................................</td>
</tr>
<tr>
<td>3 Middleware platforms ...................................................</td>
</tr>
<tr>
<td>3.1 Definitions ............................................................</td>
</tr>
<tr>
<td>3.2 Market and customers needs .......................................</td>
</tr>
<tr>
<td>3.3 Vendible components ................................................</td>
</tr>
<tr>
<td>4 Selecting pricing model for middleware platform .................</td>
</tr>
<tr>
<td>4.1 Value of the middleware platform ..................................</td>
</tr>
<tr>
<td>4.2 Suitable licensing models for middleware platform ..........</td>
</tr>
<tr>
<td>4.3 Middleware platform pricing example .........................</td>
</tr>
<tr>
<td>4.3.1 Modules ....................................................................</td>
</tr>
<tr>
<td>4.3.2 Developer tools ....................................................</td>
</tr>
<tr>
<td>5 Conclusions .................................................................</td>
</tr>
<tr>
<td>References .......................................................................</td>
</tr>
</tbody>
</table>
1 Introduction

Software has an important role in most business areas today. Without software many processes would be more time consuming, or sometimes even impossible. Therefore the value of any software should not be underestimated. Pricing the software right should make it possible to return some of this value back to the developer.

Software development is not an easy process, and the functional requirements and performance expectations for the software tend to grow all the time. To fulfill these requirements and expectations, a large group of software specialists is often needed to design complex software solutions. In addition an extensive testing process is also needed, because of the high software performance and scalability requirements.

Even though the importance of any software would be known, defining the price for it is never straightforward. When the software is ready, copying it to multiple CDs or other media does not require much effort, and causes only marginal costs. However, the intellectual capital used for producing the software should be somehow refunded to the software provider. Pricing a software platform is even more challenging as the buyer of the software platform is normally not the end user of the software. This adds one more step to the value chain and thus complicates the evaluation of the value of the software.

In addition, pricing is one of the variables that can influence the success or failure of the product. Setting the initial price too high to cover the existing development and testing costs quickly might drive away too many potential customers. But on the other hand too low a price will cause lost revenue opportunities to the vendor, and complicate the further development and testing process. Therefore it is important to plan the pricing structure of any software product well in advance. For that the knowledge of the business area where the software will be used is essential.

The objective of this paper is to find a suitable pricing method for middleware platforms that can be considered as a one type of a platform. In Chapter 2 some of the most common licensing methods are presented. Chapter 3 then gives a definition to middleware platform and explains its main components from the pricing point of view. In chapter 4 the licensing alternatives are evaluated against the characteristics of a middleware platform. And, finally Chapter 5 concludes the study.
2 Software pricing methods

There are no official models or templates for software pricing. Every company has its own way for pricing software, and some best practices have evolved over time. Initially, the pricing methods were based on the processing power of the mainframes as a function of their central processing unit, as the software was only run on mainframe environments. The first companies purchasing software were banks and large corporations, so it was expected that the software would be used continuously, and therefore the customer with a faster CPU would have to pay more. [1]

2.1 Value of the software

Software development software can be expensive and time consuming, but once the software exists, the costs of copying it to multiple customers are small; basically the cost of the medium (e.g. CD) the software is delivered on. This means that the price for the software cannot be derived from the material expenses, as the biggest cost is generated by the intellectual capital of the company, the developers. There does not either exist any official measurement tools for defining the value of the software. These two facts make software pricing a challenge. [1]

The best starting point for defining a suitable price for software is to try to understand the value it provides to its users, so to customers of the software vendor. The main value of the software is the functionality that it provides and thus the revenue increase or cost reduction to the customer through it. The cost reduction can be achieved e.g. with reduced complexity in processes or system architecture, reduction in maintenance costs, or reduction in the total cost of ownership (TCO).

Identifying software’s value to the customer is not easy. It requires extensive knowledge of the business area and also competitor analysis. When the value of the software to the customer is know, or at least estimated, the price can then be aligned according to benefits the customer is expected to receive from it.

Software, when sold, provides value also to the vendor. The main value is naturally the revenue received from the software sales, but there can also be other kinds of values that the vendor wants to consider when deciding on the price. In some cases the vendor might wish to sell the software with lower price than what the value
proposition to the customer would allow to increase its market share and user base. Or the vendor might wish to give additional functionality free of charge for some period of time to increase up-sell to the advanced version of the software.[1]

2.2 Licensing

Licenses are today used in most cases when selling software. Because software can be copied very easily, software vendors prefer to sell the right to use the software in certain restricted ways (i.e. the license), rather than just giving out the software on the CD to be used and copied freely.

A license, according to Chávez et al., enables an entity (licensor) that owns the property rights in something to grant to a third party (the licensee) the right to use those property rights [2]. Ferrante defines a software license as any procedure that lets an enterprise or user purchase, install, and use software on a machine or network in accordance with software vendor’s licensing agreement [3].

A license should contain at least the rights the licensor authorizes the licensee to exercise in the software and payments to the licensor. In addition it should define the risks and liability each party assumes under the license, the support, maintenance and warranties for the software, the confidentiality of the licensed component [2], and the term of the license, i.e. when the license begins and how long it will last [4].

Other specifications a license can include vary depending on the product. The license can contain restrictions for software use, and also geographical restrictions are possible. Special cases of a software license are so called source code licenses. These licenses give an additional entitlement to the user to see and modify the source code. [4]

2.3 Used licensing methods

No official licensing methods exist, as was already mentioned earlier. This Chapter presents some of the most popular ways software vendors use for licensing their products. Some of the models are alternatives while some of them can be combined.
2.3.1 Packaged perpetual licenses

This is probably the most common traditional licensing model for software that is sold to individual users. It is based either on locking or on key expiration mechanism to protect the software. The locking mechanism can vary from machine host name, Ethernet MAC address, IP address to the use of dongle. A dongle is a hardware device which is plugged into parallel of USB port of the computer that is using the licensed software. This means that the license is computer specific, and changing it to be available on another computer is time and effort consuming. So, it is not user specific.

If a key expiration mechanism is used, special activation wizard software is included in the actual software. During the software installation the activation wizard requests a serial number or activation code, which allows user to continue with the installation if entered correctly. Some software allow user to use the software for free for some trial period, e.g. 30 or 60 days. When the trial period is over, the activation wizard requests the specified code, before allowing further usage of the software.

Packaged perpetual licenses are not suitable for software that is targeted to enterprises with multiple users. Maintaining the record of existing users in a company with thousands of employees, and where multiple employees arrive or leave the company continuously is very challenging. Therefore this licensing model is more suitable for single users or teams with a small number of users. [3]

2.3.2 Server based licenses

Server based licensing is also a traditional model. In this licensing model, the number of processors (central processing units, CPUs) running the software defines the number of licenses purchased. This model is very close to first software pricing method that was based on the processing power of the mainframes. This model has been used in software that is normally run continuously in large centralized server systems, like e.g. in many software products from Oracle, IBM or Microsoft.

However, when multicore processing (i.e. having multiple processors on a single chip) emerged the market, this licensing model had to be redefined. The most common alternatives used today are counting the sockets (i.e. the number or connections) instead of the chips or counting by each core regardless of the number of chips. [3]
2.3.3 Network licenses

Network licenses, also known as user licenses, are popular in large companies, as they enable easy deployment and license management of thousands of copies of the software products to multiple users. This model is based on server applications that manage the software license distribution. [3]

Network licenses can be divided into two categories: the named licenses or to concurrent (floating) licenses. The first one allocates the software license to specified (named) users, and normally does not contain any other usage-limits. It is also the cheaper alternative, but does not provide much flexibility. [1]

The idea of concurrent licenses is to provide a pool, a fixed number of licenses to a group of users. This provides flexibility, as the licenses can be allocated to any of the users when needed and freed again when either a specified license borrowing time expires or the user stops using the software. In most cases the number of licenses is smaller than the number of potential users in the group. The idea is that not all users need the software continuously and at the same time. However, the company must make good estimates of the number of possible concurrent user to be able to find the right number of licenses needed for the pool. [1], [3].

In addition, some software vendors also provide differently priced user licenses to heavy users and light users, where the first has access to all functionality while the latter only to some specified functionality like reports etc.

2.3.4 Unbundled licensing

Unbundled licensing model breaks the software into a base component and optional components. Each optional component contains some additional functionality area. This is often preferred licensing model today, as the software development has become more module-oriented. It allows the customer to select and pay only for such parts of the product it really is planning to use and it also enables the vendor future sales opportunities for the existing customers, as new optional components are developed. [1]
2.3.5 Risk-sharing licensing

Risk-sharing licensing can be used when the customer and/or the vendor is not sure how much value can be expected from the purchased license. The value can be related to the resale of the software inside customer’s own solution, or to the changes of the customer’s business size. [1]

The basic idea of this licensing model is that customer pays first a basic payment for the product, which is often not a very large sum. When the customer starts receiving more value from the usage of the product, either as revenue from the resale or other clear business benefits, then some fixed part of that value is transferred also to the software vendor.

This model is especially popular with software components, where the customer develops own solution containing the vendor’s software component. When the customer then sells the final solution to the end user, the software component vendor receives a certain percentage of each solution sold.[2]

This licensing model reduces the initial investment and the risk of the customer, and increases the risk of the vendor. However, it can also provide very stable income for the vendor if the customer is able to sell its final solution well, and so enables well the price of the software to follow its value to the customer.[2]
3 Middleware platforms

Terms *Middleware* and *Platform* are used quite freely in the computing literature. This Section presents first high level definitions for these terms, which seem to apply to most cases. Then an overview to the middleware platform customer segments and market is given. Finally this Chapter lists the most common components of a middleware platform from the sales point of view. So, it presents such sellable components that can be priced separately.

3.1 Definitions

Wikipedia defines (software) platform as a framework, which allows other software to run on, and middleware as software that sits between applications, that are working on different operating systems [7], [6]. Other similar definitions to middleware provide e.g. the glossary from BBN technologies, where middleware is defined as any program that mediates between two separate and usually already existing programs.[8]

Taudes et al. [5] define software platform as a software package that enables the realization of application systems. They also list some examples of software platforms like operating systems, database systems, workflow systems and customizable application packages like SAP.

Blair et al. [9] define middleware’s role in software architecture as to present a unified programming model to application writers and to mask out problems of heterogeneity and distribution.

Based on the definitions above, this work defines middleware platform as a software platform which enables the development of a middleware-product. The final middleware-product is needed for hiding the complexity of the applications from other applications, by providing standardized interfaces to both directions.
3.2 Market and customers needs

The demand of commercial-of-the-shelf (COTS) software components has increased as the software systems are growing larger, more complex and more independent [12]. Platforms can be seen as a kind of software component and the business reasons for using them in software development are similar to the reasons for using any COTS software component [10]:

- Reduced total development costs as some generic functionality already exists in the platform/software component.

- Better system quality and performance, as the testing of the platform/component is done by the vendor, and the vendor receives feedback from multiple customers.

- Shorter time to market, as part of the final solution already exists.

Similarly the need for different kinds of middleware solutions is increasing, as the overall IT-system architectures in most companies tend to grow and become more complex over time. Integrating multiple software systems with varying interfaces is very expensive, and every time any of the systems change, the integrations have to be renewed. Middleware provides with standardized interfaces a communication media for the existing systems, and so reduces the integration need (see Figures 1, 2).

![IT system architecture without middleware](image_url)

Figure 1 IT system architecture without middleware
Figure 2  IT system architecture when middleware is used

Middleware platforms are not ready solutions for end customers, but a framework for middleware developers, who can develop their own specific middleware solution on top of the platform. So, the customers of the middleware platform vendor are other software vendors.

In order to understand the whole market, the platform vendors need also to be aware of the possible customer base of their customers. So, e.g. what end-customer needs can be satisfied with a product that is developed on top of the platform. The more flexible and generic the platform is, the wider the end-customer base can be as well as the revenue opportunities for both to the middleware platform provider as to the middleware solution provider.

### 3.3 Vendible components

In software platform there exists normally a core module that is the mandatory part of the software. In the middleware platform case this core can be a very simplified module, providing only very basic functionality like user management, modest middleware parameterization capabilities, few application programming interfaces (API), and maybe some standardized interfaces. The core part can be used as such for the middleware development, but in most cases also additional functionality is purchased with the core. [9]

Interfaces are one of the most important parts of any middleware solution, as the target of the middleware is to be in between of multiple applications and hide each individual application’s complexity from the other applications. A Middleware solution has to provide flexible and extensive interfacing capability (so called Application Programming Interfaces, APIs) to its services to enable integration with...
complex systems. Also ready-made standard interfaces like web service interfaces or file import/export interfaces are needed to fulfill the needs of the customer. [11], [6], [13]. As mentioned earlier, some of the interfaces can be part of the core middleware platform. However, as developing interfaces can be very time and effort consuming, it is common that at least some complex interfaces are sold separately, as additional modules to the core module.

Even though the minimum required functionality is already available in the core module of the middleware platform, vendors often offer also additional functional modules. The motivation for these additional functional modules from the customer point of view is the same as the motivation of using any software component: saving time and development effort, and ensuring better quality.

Depending on the complexity of the middleware platform and on the complexity of the possible interface needs, the middleware platform can also contain developer tools. These tools can be used e.g. for creating such interfaces that are not provided by the platform vendor, configuring and customizing the final solution, or even developing new functionality on top of the middleware platform.

Figure 3 below presents one possible composition of a middleware platform, containing the components presented earlier in this Chapter.

![Middleware platform components](image-url)
4 Selecting pricing model for middleware platform

Based on the common components of the middleware platform, and the most common software licensing methods presented in the previous Section, this Section evaluates the possible pricing models for a middleware platform presented in Figure 3.

4.1 Value of the middleware platform

Software platforms in general do not directly generate value to their users (customers), but they enable the implementation of various applications that can then be sold to the end customers. Thus the value of the platform materializes itself when the final product containing the platform is sold. This makes it hard to define the price of the platform as the actual final value to the user (the developer) might not be known in advance. [5]

The possible customer base of the final solution might not be known initially. The core module of the platform can enable the development of many different applications for different end-customer segments, and thus provide many revenue sources for the developer.

The additional functionality modules increase the value of the final solution directly, but might not be suitable to all end-customer segments that the developer is targeting. So their value is not as directly related to all possible customer segments as the value of the core module is.

The value of the interfaces that the platform offers out-of-the-box can often be seen similar as the value of the additional functionality modules. However, for the middleware platform this is not the case. The target of the final middleware product is to hide the complexity of the systems above or below it, by offering the standardized interfaces. Therefore the more standardized interfaces the middleware can offer the more customers its provider is likely to gain. Developing interfaces is often time and effort consuming and requires extensive testing, so the more ready-made interfaces the middleware platform provides, the bigger the value to the developer.
The value of the developer tools to the developer differ from the value of the other middleware platform components. In most cases the developing tools are not sold further to the end-customer, so the value of the developer tools realizes itself to the developer with shorter development times and better solution quality.

### 4.2 Suitable licensing models for middleware platform

Basically all licensing models presented in the Chapter 2 can be used when pricing middleware platform. However, some of them are not suitable because using them would not reflect the value of the platform to the developer back to the platform provider. Pricing of the developer tools will be discussed here separately as their value calculation varies significantly from the value calculation of the other modules.

Packaged perpetual licenses are not suitable as such for middleware platform pricing. Handing over IP-addresses from the end-customers to the platform vendor or forwarding multiple key-codes from the platform vendor to the developer and then to the end-customers is not convenient. And, having only one price for the whole platform would either increase the price too high for the developer, or would not be related to the value the developer finally receives from the platform.

Also server based licensing method cannot be utilized solely in middleware platform pricing. Defining the number of servers which the platform based product will be running on does not reflect the possible revenue opportunity of the developer, if the final product is used by multiple end customers. Network licensing, so defining number of users who can use the platform has similar problems as the server based licensing. Defining the number of the developers developing the final solution does not reflect the potential number of customers of the developer.

In both server and network licensing models the price could, however, be calculated based on the expected number of end-customers’ servers or users. This, nevertheless, would not be very convenient for two reasons. Firstly, defining the potential number of end-customers at the instance when the platform is sold to the developer is very difficult and might lead to much distorted pricing. Secondly in cases where the potential end-customer base is expected to be very large, the price of the platform would be too high for the developer.

Unbundled licensing model suits well for middleware platforms. As Figure 3 presents, the middleware platform product can be divided into multiple sellable items
that can be priced separately. However, the core module must be mandatory part of any module composition of the product.

Risk-sharing licensing is also suitable for products like middleware platform. The developer would initially pay only a relative small price for the platform, but would then transfer some percentage or part of the income received from the final product to the platform vendor.

For developer tools the risk-sharing license is not suitable in most cases. The traditional packaged perpetual licensing or network licensing might be the best alternatives. The former being more suitable for simple developer tools used only by a small developer team with only modest added value through reduced development times. And the latter being more suitable for extensive tools that can be used by many people in the developer’s organization. Unbundling model can be also used for developer tools, if they consist of different components that can be purchased separately.

4.3 Middleware platform pricing example

Unbundled licensing model can be seen as a starting point for creating an optimal licensing model for a middleware platform. The platform consists of multiple components, so each component should be priced separately.

4.3.1 Modules

Risk-sharing licensing should be the main tool for the middleware platform licensing. The modifiable components in the model are

- Initial payment for the core module (ic), sometimes also called as developer license.
- Initial payments for the possible additional functionality modules (ia)
- Initial payments for the possible additional interfaces (ii)
- Payments related to size the final product sales (fs). This can be called run-time license
The total price ($tp$) during the platform lifecycle is then formed as follows (excluding developer tools):

$$tp = ic + \sum_n i a_n + \sum_m i i_m + \sum_p k_p (fs_p)$$

The size of the initial payment for the core module ($ic$) depends on the maturity and reliability of the platform. If the platform is mature and already used by many customers, the initial price can be higher, as the usefulness of the platform has already been proven. Also the development effort of the platform and possible richness in the functionality can increase the price, as the platform-vendor hopes to receive some revenue from the product already in short term. In the case of a new platform with no reference users yet, it might though in some cases be beneficial to the vendor to offer the core module for free ($ic = 0$) to the developer.

The initial payment of the additional functionality modules and the interfaces should be in line with the initial payment of the core module. Naturally the price of the more complex interface or functionality module should be higher than the price of a simple additional module. In the middleware case especially the interface pricing should be considered carefully, to ensure that the price of special interfaces stays high enough.

Run-time licenses can be defined in various ways. The refund price to the platform vendor ($fs$) can be fixed and so multiplied simply by the number ($k$) of the sold products. Or it can be related to the size of each deal the developer makes with the end-customers. The size, again, can be defined in various ways: it can be related to the system size ($k$) at the end-customer (server licensing), to the number ($k$) of simultaneous users (network licensing) or simply be a fixed percentage ($k$) of the price ($fs$) the end-customer is paying to the developer.

Even though server licensing and network licensing are possible in this case, the simplest, and so the best solution is to use the percentage from the revenue the developer receives from each deal. However, in case the developer does not want to reveal the deal sizes, then the network or server licensing can be used, or a fixed sum per deal can be defined.

### 4.3.2 Developer tools

As was mentioned earlier, risk-sharing is not a suitable alternative for developer tools. Network, server or packaged perpetual licensing models can be used, but it is
also common that the platform vendor provides the developer tools free of charge with the core module.

In case the developer tools are very sophisticated with extensive functionality, the vendor might consider dividing the tools into two categories: to basic tools which are free of charge, and to enhanced developer tools which can be purchased for a certain price. If the middleware platform’s developer tools include tools for creating new interfaces easily, these tools should be part of the enhanced developer tool set.

## 5 Conclusions

Exiting software pricing methods can be used for pricing middleware platforms. The biggest challenge in pricing is to estimate the actual value the developer will have from the middleware platform. The more specialized the platform is, the better the platform vendor can estimate the possible use cases and the possible customer base of the products developed on top of the platform.

In most cases it is not possible to estimate in advance the number of end customers the developer will have, and thus the actual value of the platform. Therefore the safest way for the middleware platform vendor is to use risk-sharing licensing, varied in a suitable way. This will ensure that the value the developer will receive from the final product will be transferred to a certain extent to the platform vendor. However, it also means that the platform vendor will not receive compensation from its product in one go.

Because software development is not cheap and the platform vendor will need to be able to finance also further software platform development activities, an initial payment at the time of purchase could be required. The size of the initial payment has to be in line with the reliability and usefulness of the platform. New emerging platform vendors might need to give the platform to the developers free of charge and wait patiently for the developer’s final product sales, while vendors with mature platform product and/or existing customer base can request higher initial payment.

Pricing additional functionality modules should follow the pricing of the core module of the platform. However, as the interfaces can be seen as critical parts of any middleware product, their pricing should be fine tuned with additional care.
Developer tools can be priced in a more straightforward way with traditional licensing methods, e.g. by using the number of developers (networking license) as a pricing parameter, if they are not given for free with the core module.
References


