Cost comparisons between open-source and proprietary software

Dzhan Dimitrov - 11808940

Date: December 17, 2020

Contents

I hereby declare that:

1. I have written this paper myself, independently and without the aid of unfair or unauthorized resources. Whenever content has been taken directly or indirectly from other sources, this has been indicated and the source referenced.

2. This paper has not previously been presented as an examination paper in this or any other form in Austria or abroad.

3. This paper is identical with the thesis assessed by the examiner.

[illegible]

Date: December 17, 2020

1 Introductiont

In this seminal work we examine the cost structure of open-source and proprietary software, perceiving software development as an ongoing investment activity. This software decision-making process can be viewed as the expenditure of valuable resources (such as time, talent, money) with the end goal of creating surplus value. We include insight about the design and strategy decisions for increased productivity of the software investments and propose more sophisticated method for evaluating projects. In the first section we present insights for software economics and subtle aspects for evaluating the cost/return characteristics of software projects. Because we examine the costs as an investment of valuable resources, the seminal work also puts an emphasis on different sources of value that can justify the invested resources. In this mean we consider more enhanced notion for costs and we also include the missed sources of value (opportunity costs), instead of simply considering the expenditure of material resources on a given project. We also provide models for more comprehensive cost estimation and provide some insights of the software economics.

Section 2 and 3 provide models for cost estimation and better design of software development projects, as well as the different kinds of costs that every project face by developing and marketing a software product. Some categories of open-source and proprietary software and their differences in licensing structure are clarified. These differences have some advantages and some disadvantages, that are further examined. They become more distinguishable after reading section 5, where we show some case studies of open-source and proprietary companies, their strategies, and their costs (particularly on research and development) by software development.

We also examined the motivation of developers to participate in open source and showed some reasonable advantages that comes along the participation in open source. We referred also to altruistic and self-fulfillment motives. These can be read in section 4.

2 The economics behind software development

The sustainable advancement of computer science enables the creation of additional utility for all kinds of systems. The value creation consists in information processing in high speeds and low costs and changes processes in all industries. Although the hardware is the enabler for this change, it is the software that represents the value creation. Information technology spending on enterprise worldwide for 2019 is 477 billion U.S. dollars (Statista, 2020).

In this large industry some losses are inevitable because of the effective working marketplace. But sometimes the consequences of non-repellent software product are too large and unpredictable. It is important to have clear vision regarding the risk-return characteristics by the development of software products. In this section we will examine some problems in software development regardless if the project is open source or proprietary. This will allow us to become more acquainted with the economics behind the software development before starting any comparisons. In both cases software development is a matter of procreation of valuable resources for the participating stakeholders, that can be volunteer developer communities or business managers. The values put in this investment can be time and talent. As the cost consists mostly of effort, which we can translate to monetary terms. Some of the benefits in return can be non-monetary as we will discuss in further section.
The sustainable advancement of computer science enables the creation of additional utility for all kinds of systems. With the development of the IT-industry, the work efficiency has grown enormously, and the costs have sunk. Although the hardware is the enabler for this change, it is the software that represents the value creation. Information technology spending on enterprise worldwide for 2019 is 477 billion U.S. dollars (Statista, 2020).

Not all software projects can be successful, but sometimes the consequences of non-repellent software product are too large and unpredictable. It is important to have clear vision to the risk-return characteristics by the development of software products. In this section we will examine some problems in software development. This will allow us to become more acquainted with the economics behind the software development before starting any comparisons. In both cases it is a matter of procreation of valuable resources for the participating stakeholders, as volunteer developer communities, business managers or users.

2.1 The logic of successful software

In businesses today investing in software is a central aspect. To understand the economics of software development we must look to the matter as expenditure of valuable resources anticipating future returns. What we want is the greatest possible return. To assess it in monetary terms we can use axioms of corporate finance. The value today is assessed by the expected future gains. For project to be successful, it must produce more benefits than the used resources and in less costs that the competitors need to produce the same value. Or to produce more value than competitor at equal cost (Barry W. Boehm and Kevin J. Sullivan, 1999). A blunder that may appear is considering only technical parts of the work. There should always be link between the software and the created value. The logic of good software design is a logic of value creation (Barry W. Boehm and Kevin J. Sullivan, 1999). Another good way of considering this matter is in the lights of successful philanthropic foundations. Clever philanthropic foundations have greater impact on social benefits at the same cost or create the same social benefits at equal cost.

Because of the work and department separation in the corporate world it may be that sometimes this link between software development and value creation gets lost. The developers are guided from technical perspective. The correct models for them are these of mathematical abstraction and that can lead to remoteness to the enterprise obligation for value delivery. This problem may be deeper than it appears to be. Because these connections are unclear also for senior management, which finds difficult to track how investments at technical level helps for value delivery (Barry W. Boehm and Kevin J. Sullivan, 1999). In their article authors Barry Boehm and Kevin J. Sullivan argue that there is a lack of adequate modelling that can measure the connections between technical decision and value creation (Barry W. Boehm and Kevin J. Sullivan, 1999). They suggest that the firms should be able to account on their software projects as capital investments, which means to be able to involve third parties, selling them project risks and warranties. As a result of that some shortcomings behind software economics can be overcomed.

2.2 Subtle aspects of software economics, that must be included in cost/value calculations

Here we will reflect on some aspects in software economics may remain subtle by evaluating the software and making cost-value calculations. Sometimes the developed product may have a great potential that can be triggered by uncertain circumstances. Thus, it is important to have a strategic concern about the possible future synergies and additional costs for faster shipment of the product.

2.2.1 Considering the scalability of software as a value source

As we consider the value of a company, we do not look only at its current revenue streams. One very important aspect is to consider what opportunities may the company have regarding its current position in the market. If the company does not make any revenue, it does not exclude the possibility that it won't make revenues in the future. We must consider if a company can eventually is in good position to exploit opportunities when they arise (Trigeorgis, 1997). This was the case with Amazon. Jeff Bezos, the owner of the company, was named Man of the year for 1999 by Times Magazine, when the company was selling only books (Barry W. Boehm and Kevin J. Sullivan, 1999). Back then the business they were running was not in the scale that is today, but they had the synergies and options ahead of them because of their position in the market. This way of thinking applies also for software, especially for open-source software.

It is important for developers to understand how they can increase the value of the product they develop by designing it in sustainable way, which create advantages for the future. These decisions regarding the design of the product must be constructed in strategic way. In competitive markets the options one has for the future can create a big difference regarding the position of the product or company in the future. All these factors must be considered while investing funds into a project. When it comes to return, they may not occur or may require complex economic drivers to be from any relevance. So how can we measure the value of such matter.

One way to measure the utility of this advantage is to find an option price. This method provides forecast for the probabilistic future gains. Another advantage of pricing current software project as an option is that it does not rely on subjective estimation but existing data from the financial markets (Barry W. Boehm and Kevin J. Sullivan, 1999). The essence is to estimate the price considering the fluctuation of the value over time. That will give the estimated value of the property over a given timeframe as a function of the volatility. The results are objective because they derive from the market as the comparable assets must have the same amount of risk. Finally, we must evaluate if the value of the option is worth the initial investment to obtain the architecture, which enables the exploitation of the option.

As we speak for value, it is also important to mention that value does not have to always be monetary. Sometimes it even cannot be measured as a scalar quantity. Let us consider the indifferent position between cost and safety. In this situation every tiny measure of safety can be more valuable than cost, which makes it ineffectual for scientific comparisons (Haimes, 1999). It is interesting that the greatest and the longest enduring companies do not make money their highest priority. Instead, they see it as intermediary for creating value in other dimensions (Collins; Porras, 1997).

2.2.2 Considering late delivery time of software as opportunity cost

Another aspect that has the be considered is the timeframe when the software is delivered and how the number of developers working on project influence the project duration. In this term the proprietary software often has an advantage in terms of the ability to plan the time when the software is complete. The trade-off here consists in the opportunity costs on the one hand and the lower efficiency on the other when more developers are put in work. In the literature more ways for the estimation of the cost of software compared to its benefits can be found, such as development cycle time, delivered quality, synergies to other software, and options for strategic opportunities as discussed (Barry W. Boehm and Kevin J. Sullivan, 1999). For value-based investments it is crucial to be able to assess the delivered value. Thus, the endeavour put on the assessment of the costs in unity with the value of the created product (which is linked with uncertainties, competition, and incomplete information) is critical for the investment decisions for enterprises (Barry W. Boehm and Kevin J. Sullivan, 1999).

An example for models that estimates the most cost-efficient time-frame for developing software is the rule of thumb:

$$\texttt{Calendar Months} = 3 \cdot \sqrt[3]{\texttt{Person-Months|}}$$

It states that for example in a 27 person-month project the most efficient schedule would be 9 months for group of 3 people. The drawback of this function is that it considers only the direct cost for the development of the project and it neglects the opportunity cost for delivering the product in the marketplace before the competitors. These considerations are decisive for creating advantage in today's world. Thus, while considering costs in the development of open source or proprietary software we must also look from a broader perspective.

The software product can be viewed as an option in a stock market. If after the development, the software happens to have a value it is shipped to the market. Or, consequently, if after completion of a phase in the software development it is reasonable in the lights of the new information and circumstances the next phase is started. That is the equivalent of exercising the option. If a competitor enters the market and gains a portion of the cash flow, that can be viewed as a drop in the dividends and hence the stock price. Dividends are the big motives for competitors to enter the market. The catch in this situation is that the more lucrative product produced by the company (pays a lot of dividends), the more will be the incentive for concurrents to enter the market. The shipment of the product to the market is crucial because it means that the company will be able to skim the market for a given period (until competitors enter the market too). That should be considered, when planning the time to release the product and will have direct reflection over design-decisions over the product. Since the time-to-market is any delay is connected to opportunity costs. For example, time-to-market can be crucial for government contracts (Barry W. Boehm and Kevin J. Sullivan, 1999).

This gives a significant advantage to proprietary software in comparison with open-source software since the implementation schedules in open source are not tight or they lack deadlines. The loss of coordination because of the scattered community causes more delay, and some opportunities are sunken. In comparison in proprietary firms the developers have timely objectives, which leads to better results, at least economically.

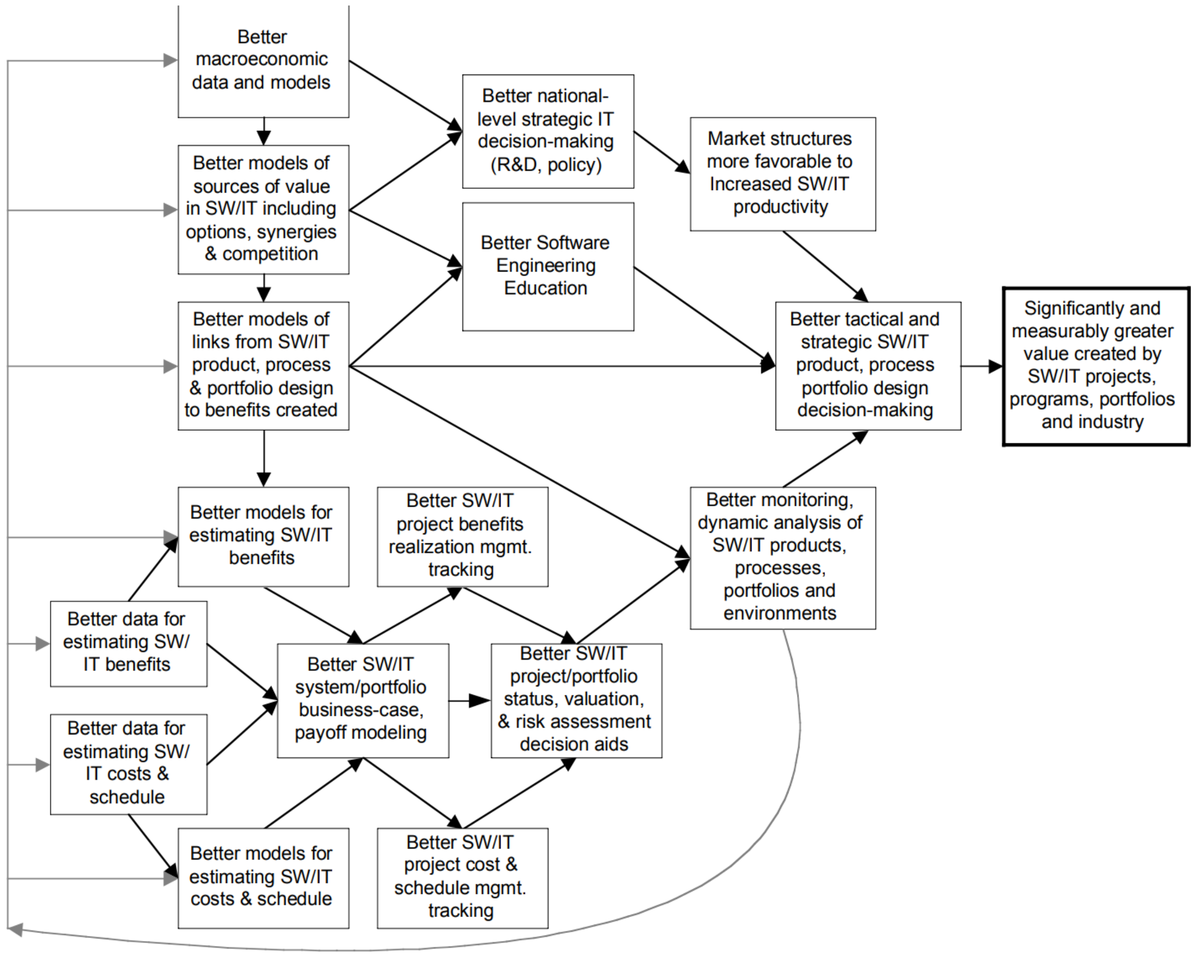

Better macroeconomic data and models
Better models of sources of value in SW/IT including options, synergies & competition
Better models of links from SW/IT product, process & portfolio design to benefits created
Better national-level strategic IT decision-making (R&D, policy)
Better Software Engineering Education
Market structures more favorable to Increased SW/IT productivity
Better tactical and strategic SW/IT product, process portfolio design decision-making
Significantly and measurably greater value created by SW/IT projects, programs, portfolios and industry
Better models for estimating SW/IT benefits
Better SW/IT project benefits realization mgmt. tracking
Better monitoring, dynamic analysis of SW/IT products, processes, portfolios and environments
Better data for estimating SW/IT benefits
Better SW/IT system/portfolio business-case, payoff modeling
Better SW/IT project/portfolio status, valuation, & risk assessment decision aids
Better data for estimating SW/IT costs & schedule
Better models for estimating SW/IT costs & schedule
Better SW/IT project cost & schedule mgmt. tracking

Image source (Barry W. Boehm and Kevin J. Sullivan, 1999).

To estimate the costs of software development project we need not only the estimation model for the direct cost connected to the project but also business operational mission domains in to be able to make trade-offs between development time, effort put in work, quality, functional diversity, and the opportunity to create a real value for existing demand (Barry W. Boehm and Kevin J. Sullivan, 1999). The aim by comparing costs by software development must be to have more broad, dynamic, and strategic overlook to the matter. This will allow more precise decision making by reflecting on different circumstances.

2.3 How can the process of software creation be improved?

The roadmap of Barry W. Boehm and Kevin J. Sullivan is developed to build a frame, which will enable measurable increase in the delivered value of IT-projects. This may appear straightforward to the reader but there are subtleties, which must be considered to construct a proper way for project evaluation. For example, in business context the present value is calculated differentiating future options, whose future values must be discounted to the present moment and we should include the possible exploitation of additional markets in our evaluation. These options may have a great value, that are not realized, and it can lead to abandoning of promising projects. In their work Barry W. Boehm and Kevin J. Sullivan emphasize that value is a complex quality, which can be unapparent at the first sight. So, evaluating the projects in terms of the realized potential, it is crucial to have methods for their better evaluation.

The Roadmap in the figure above pictures backwards a network of important outcomes and intermediate outcomes. The interdependencies of the outcomes can be tracked and there are provided important paths, how the models and methods can improve. We can distinguish two parts in this roadmap. In the lower part the roadmap handles tactical matters (for example how can the costs of the project be estimated more accurately). The parts above are linked to strategic concerns (for example which options emerge by developing the project and what synergies can be used).

2.3.1 Flat decision-making for better value delivery

One outcome of the proposed roadmap is that designers at all levels must take part in determining the structure of larger programs and participate in process design to enable better value delivery. As a result of that the management of the project becomes more dynamic and programs can be distributed strategically in a portfolio (Barry W. Boehm and Kevin J. Sullivan, 1999).

To make the proper decision there should be considered the following matters. The design space must be broad enough. The design space is determined from the market situation – the competitors and their products. Influence on the structure have variety of conditions – for example, the national-level strategy for long term research (where the product is being developed) and development expenses. So, these external factors determine what materials are produced, that can be included in our design space and which properties they have.

Another issue that is covered by the roadmap is the need for better links between technical design, current circumstances, and value creation. This can be achieved by better models of the sources of value. These models can be used by the developers and will result in better decisions. One example of such a link can be the increased price of an option in a volatile market situation.

So that the roadmap can function the people in management must have deeper understanding of the technical part of the work. They personally must understand the sources of value and how they can be transformed in a value creating manner. This is a crucial prerequisite for combining technical and strategic decisions to catch the greatest possible value.

There is a need for dynamic monitoring, which will steer the decisions through the design-space (the multidimensional combination and the interaction between the materials and processes used). This means constantly gathering information about valuation and risk of the project, including external factors like price of the materials, macroeconomic climate, moves of competitors. The result is a dynamic and more correct evaluation of the possible cost-benefit dependencies. In essence we check the relevance of the business model that we build upon our project in each step before proceeding in further phases.

The design-space in which the developers operate in IT-projects are less developed in comparison to other fields. If we take as an example the automotive industry, we see that there are specialised producers of every component of the car and they compete fiercely for the prices of their products. When we look in IT-field we see that options for buying third-party components are less developed. Furthermore, there is less chances for managing the risk of projects. Warranties and insurance are common in all industries. An orange producer can buy future for oranges and ensure his business if the price of oranges exceeds given amount. The lack of opportunities to manage risk with market-based mechanisms is an obstacle for the efficient project development.

2.3.2 Creating direct links between technical parameters and business aspects

While creating a software a lot of technical and managerial decisions must be made. Which formal methods will be used, what will the software architecture look like? These are common technical issues. Managerial issues are the decision about the changes in a program. Should the program continue when new information about the conjecture occurs or should be stopped. Even though it looks like these two matters are separable from each other, their links are decisive for the value, which is being created.

Important method for adapting to new situations is to use design for change. The software components must be created in a way that allows substitutability – modifying or replacing components. That will allow a prolonged life cycle of the created product, which results in higher chance to deliver more value than the spent goods/efforts. Thus, this method is seen as a value-maximizing. This rule is very reasonable, but in addition to that one must also include the circumstances in his calculations. If the design intervals are high and it takes a lot of additional time to implement a project in modualar way, while at the same time the project is highly competitive and time to market is crucial for success or failure, then it can be more advantageous to not implement this method. Other possible questions are the likelihood of change, so that the implemented method is of use, the time when the change can occur, and the importance that it will have in terms of obtained benefits. What we want to show with these circumstances, is that the decisions for software architecture goes always hand in hand with managerial concerns. Any one-sided approach taken in software development project is a failure in heuristic. It is a bad way for calculating costs or expected returns. Circumstances may impose the need for free open-source version of a software product and circumstances may change.

Software development process usually flows in uncertain environment with limited knowledge and a lot of external factors that cannot be precalculated. Often the work that a developer puts on one project is tied to the actions of competitors, the stand technology improvement, the larger project that is part of, or another macroeconomic factors. With the fluctuation of the circumstances some brilliant visions fall through, and new ones arise. It can often be the case, that the contribution of the project is to some completely different area as initially planned. The same applies for the progress of the project, which can often go through different paths that were not predicted in the initial planning. Therefore, there should always be searched for value between the lines through the evolution of the project. As we already discussed software developers envision the software design as a value creating decision-making process and understand the connections between technical and business side. They also need to implement ways to react to changing realities, so that the different areas can be dynamically monitored and controlled. These areas include the product and the related decisions for product architecture, the processes, properties, costs, risks, markets, and opportunities such as expanding the functions of the product or use synergies to enter new markets (Barry W. Boehm and Kevin J. Sullivan, 1999).

3 How to estimate the costs of a software development project?

The lower part of the diagram concentrates on how to use correct data to make a good estimation for the possible costs and benefits in the process of developing, supporting and exploiting a bundle of information technology assets. This approach has the advantage of being able to track the progress regarding cost, take dynamic managerial actions to correct the initial plan under change of circumstances and emerging of new information. In this part we concentrate on how to use relevant and up to date data for cost and benefits estimation of the project.

Examples for established models for cost estimation are COCOMO (Boehm, 1981), Estimacs (Rubin, 1985), SEER (Jensen, 1983). Problem with cost estimations with different methods is the lack of unified system for the measurement and there is a variation between different organization in the way they distinguish their data. This has led as a result that examinations within organizations are often more precise and consistent in their results. The rise of different technologies and the increase in the extent of process methodologies makes the predictions more difficult and more data points are demanded for the same amount of accuracy in comparison to the past. (Boehm, 1999) For example, the in year 2000 released version COCOMO II needs 161 data points for the same prediction accuracy in comparison to 63 data points of its predecessor, COCOMO, released in 1981. There are different approaches for estimations. All of them must cope with data, that is not precisely defined, and the rapidly changing technology.

3.1 Cost fluctuations and their interdependence with the learning curve in new projects

With the development of new projects every organization can develop a better understanding of the applications and make better forecasting for the probable costs, which will arise from the development of new software. This also leads to being able to organize the project with better estimation for how much time the project will take, and it leads also to an increase in the quality of the project planning. As the projects becomes more connate to previous ones, there can be observed a boost in the productivity. This a consequence of accumulation of knowledge and reusable components for the software. With the execution of large number of projects, the firm comes to a point, where it can develop new projects only by using the old components. That is followed by decrease in estimation error, because the teams can calculate better the related costs. This follows until the information can be recalibrated to the new circumstances. As the competence of the project members grow, they can apply packaged solutions, adapted to satisfy the needs of the purchasing organization. These are also known as COTS (commercial off-the-shelf) components. As the domain of understanding grows with time and the project members can use very high-level languages (VHLL) and develop system of systems for different domains. This is shown in the graphic below.

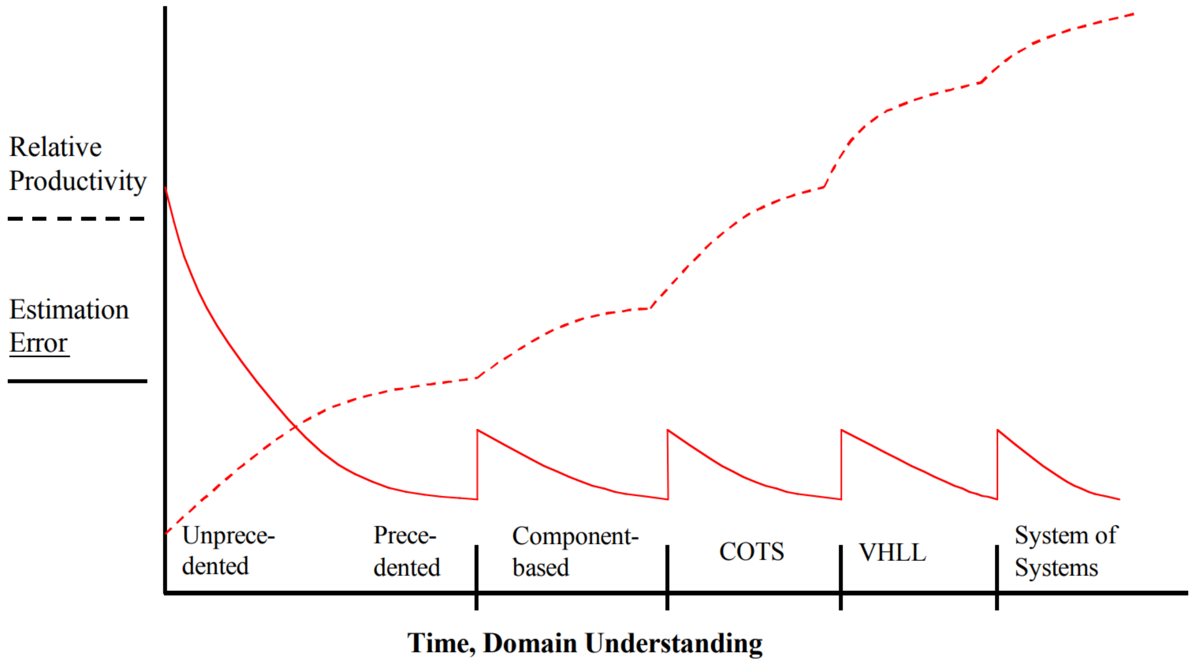


Image source (Barry W. Boehm and Kevin J. Sullivan, 1999).

3.2 Lifecycle of the designed products

Software-development is a long-term investment and a process, where a lot of cross organizational decisions are made. The aim by the development of the project is to create an additional value. Other targets like the safety, quality or user-friendliness of the product are secondary. The technical methods can be viewed as mediums for reaching the end goal.

For creating added value, the software development process must be considered as an investment action. The project must be managed and modelled in an according way. Structuring portfolio of products and creating strategy for exploitation of current or possible opportunities in the future, considering the flow of the market and the actions of another companies (Barry W. Boehm and Kevin J. Sullivan, 1999). Methods for evaluating of present value can be taken from finance. With the help of these methods, we can compare the future costs and potential benefits of a project, evaluate, and determine the tolerable risk levels, create framework that helps for better decisions, and make strategies concerning the qualities that the project must possess (for example encouraging the speed of project).

In the literature are present work for management of projects under unclarity. Breaking down the project into modules, which allows to stop the projects, if the circumstances develop poorly or create inheritor projects, if the future turns favourable and brings new opportunities. This action be spending resources for the delay of the decisions and doing so winning time and actual information.

It is natural to derive knowledge from different fields such as finance in the software engineering. The knowledge derived from finance helps software engineers to take more effective approaches. It provides new ways of making sense of product development in considering complications, risk, disinformation, and actions of competitors.

3.3 Designing the products considering future circumstances

Looking from the perspective of finance, we become sights that are not considered in the software development side. It is possible to add a new dimension to software development project, as we consider new potential sources of value. The concepts below allows us to gain new insight and become a deeper vision for possible sources for value exploitation. From the economic viewpoint catching these points increases the value of the developed project.

- What is the cash value today of future payoffs that might occur?
- What is the value of the new information that may arise during the project development process?
- Is the risk of the project tolerable and what is the risk-free value (or how less utility the project has for the developing party because of the risk)?
- What is the value of different options that depend on exogenous factors such as entry in new markets, when the circumstances are favourable; to create make complementary products after the successful entry of the developed product; to be able to abandon the current project in the occurrence of new information without losing a lot of resources; to deliver the product as soon as possible and implement market skimming strategy, when the concurrent are out of the market?
- How to distribute the assets into portfolio to reach the wanted deviation level of the portfolio?
- What are the costs of opportunity to invest early for new uncertain project that may lose value with the entrance of a new concurrent or play it safe and invest in following stages, where the returns are more certain, but less lucrative?

The chosen strategy must be pursued from the software development team. The way to implement the product according to the strategy goes with using different engineering methods. These are the following (Barry W. Boehm and Kevin J. Sullivan, 1999):

- Information hiding modularity
- Architecture first development
- Risk-based spiral development models
- The value of delaying design decisions
- Components and product-line architectures

Using modularity, the project can be divided into smaller parts. Doing so you can work on one part, while revised parts are waited to be delivered. Another advantage is that single parts can be modified and made more suitable for new market entries. The phased models for development, as the spiral model, enable intermediary decision points. That is especially useful in costly projects, where the project must be stopped as early as possible while there are not indications for success. While climbing a mountain without the necessary equipment, it is much better to return early as later.

The connection between software development models and the respective business strategies unlocks a great potential. Instead of leaving projects to be either successes or failures, they can be designed in a way that gives them a present value. Then the project has the chance to conform to the new changes. These changes either increase or decrease the value of the project. The aim is to maximize value with a flexible system that has chance to participate in future gains.

The system designer has the duty to strategically manage the portfolio with software components. The dynamic management of the portfolio is needed because of the constantly changing evaluation of the portfolio elements. If the trigger for a given element does not come, it suffers a loss or completely loses its value, because the option does not bring opportunities anymore. These triggers are exogenous and can only be predicted within probabilities. Changes can be big or small. Sometimes the actions that must be taken are affecting only parts of the system, but it can also be in a large scale. Monitoring of external factors, newly evaluating the components of the portfolio, and making strategic decisions are part of the dynamic management of the assets through software decision-making. Lastly it must be mentioned that financial reflection is a subsidiary activity that help to create better understanding for strategy. It cannot be viewed as the golden formula for successful projects. Software-development is a very complex and demanding discipline and there can not be supposed one single formula for ensuring perfection. The aim here is to produce a surplus as the created outputs ensure the stakeholders a better position and surpass the expended valuable resources.

4 Open-source software and its economic sense

Before speaking of what kinds of benefits the open-source software can offer, we must categorize the types of open source presented in this work. In this section we will relate to community open-source software and single-vendor commercial open-source software. The prior one represents most open-source projects and has different control and ownership structure.

Community open source is controlled by community of stakeholders. Examples for community open-source projects are the Linux operating system and the Apache web server (Riehle, 2012).

Single-vendor commercial open source has only one stakeholder and it serves him for a commercial exploitation. Examples for single-vendor commercial open-source projects are MySQL, SugarCRM and Alfresco.

4.1 Community open-source

Community open-source projects have meritocratic structure. They are developed from volunteer programmers. The control is determined by ownership of copyright to the code (Riehle, 2007).

Nowadays non-profit foundations are having major role in the community open source and they possess the ownership over the product. The projects are run by foundations members, who represent the community of stakeholders. The community open source can generate revenues from support and consulting services, derivative products built on the community project and increased revenue in ancillary layers of the software project (Riehle, 2007).

4.2 Single-vendor commercial open-source

To qualify as open-source firm the single-vendor open source must provide the source code of the software and the program as free and easy installable binary.

Single-vendor commercial open source do not accept contributions to the code and own the full copyright to the code and the intellectual property. They control the open-source project and create business around the open-source software. The full control over the software project is crucial, that is why these firms do not accept outside contribution without copyright transfer from the creator. However, ownership transfer can be evaded with receiving relicensing rights.

The firm can provide the software in Paid-for versions to customers under a commercial license. This dual-license strategy offers way for commercial exploitation of an open source.

4.2.1 Business function of the open source release

Once the firm releases a software and it reaches the critical level of engaged user community it can unleash a decisive effect over the different business aspects. The problem with the growing of this community are the support costs. When support is needed, only the firm, which have developed the software, can provide it. This can lead to a situation, where costs outgrow the cash flow. By putting open source on a disposal, the problem can be solved as the community becomes self-supporting. The open source accelerates the adoption of the software and avoids support costs. The single-vendor commercial firm hosts a software forge with integrated tools like forums and wikis to encourage communication between users. Users which provide feedback and solutions are rewarded. This way the community becomes self-sufficient when it comes to supporting and benefits of the resulting community are not limited to support costs, but are also connected to sales, marketing, product management and product engineering (Riehle, 2012).

Once the firm releases a software and it reaches the critical level of engaged user community it can unleash a decisive effect over the different business aspects. The problem with the growing of this community are the support costs. By putting open source on a disposal, the problem can be solved as the community becomes self-supporting. The open source accelerates the adoption of the software and avoids support costs.

The single-vendor commercial firm hosts a software forge with integrated tools like forums and wikis to encourage communication between users. Users which provide feedback and solutions are rewarded and this way the community becomes self-sufficient when it comes to support. That also reduces the support costs of the paying customers, as they often prefer to search for solutions in the community tools to save time in communication via phone calls and emails. And the benefits of the resulting community are not limited to support costs, but are also connected to sales, marketing, product management and product engineering (Riehle, 2012).

When it comes to sales, the open-source software makes it possible to track potential buyers. The potential customer will download, install, and use a product as in the illustration below. On the other hand, the firm can track the user's activity through download registration. Usually, the open source will send back usage information. That transforms the traditional pre-sales-to sale activities to a user-to-customer transition, which is a decisive benefit in terms of customer acquisition costs (Riehle 2012).

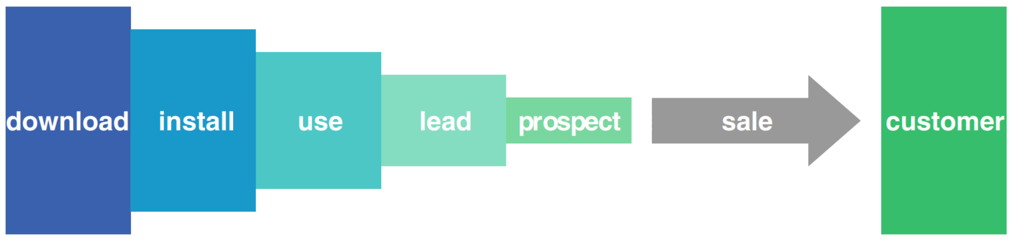


Image source (Riehle, 2012).

In setting where there is no open source the firm must engage in marketing activity to promote its product to the customer. Traditional way of doing this are advertisements and trade shows. The user community could serve as a replacement for these costs. The good testimonials of users are credible sources for effective promotion. Study shows that the Sales and Marketing costs are 2.3 times greater than research and development costs (Augustin, 2007). This creates a huge competitive advantage against competitors. The reduction of these costs could play a vital role and increase chances of commercial open-source start-ups to live up to successful times in comparison to traditional firms. In addition, if there is no open source, the customer will not be familiar with the usage of the software, hence the customer is much more likely to purchase if he is already familiar with the product. This is a huge advantage from a buyers' perspective and this setting of prior relationship can be created with open-source software.

4.2.2 How the open-source develops your product?

The users of the open-source software have their own needs and own ideas what they want from a software. Taking their suggestions and feedback is important to improve your product (von Hippel, 2005). The open source can be a big mine for product innovation for the firm. By providing open source the firms give the opportunity to the users to innovate and cocreate for free. And if the contribution consists of ideas rather than code, that is also a great deal.

The open source is a way for product managers to engage with users. That could lead to improvement of current futures and adding of future ones as well as creating a product roadmap. This will have decisive contribution to the product management. That is crucial for big software projects, because the understanding of users' needs gives the direction of product architecture. False direction in this mean can lead to large amount of sunken costs.

In addition, the observations are not limited only to the current customers, but also non-paying users, students, and researchers. This gives a broader perspective to the product managers and can help to understand different issues. For example, what keeps the non-users of becoming users or existing users to convert to customers (Riehle, 2012).

Important aspect that must be considered is the features that the open source will restrict in comparison to the paid version. Relevant restriction can annoy the open-source community and too lucrative free version can cut revenues. Thus, the product managers must decide which enterprise features are not important for the open-source community users and which features must be present in both versions. Finally, we have the advantage of community contribution in terms of product engineering. When the community is provided with the latest release of a software product it will give feedback with issues and potential bugs. Very often the open-source community would provide even the solutions like bug fix. That would not be expected at first hearing, but it is often the case from what the practitioners have experienced (MIT, 2008).

5 The motivation to be part of community open-source project

The motivation behind the participation in a community open-source project may be versatile since the work the developers put in these projects are volunteer and they do not get paid for their contribution. To put work on open-source projects the developers are driven by materialistic and non-materialistic values.

5.1 Materialistic motives in open-source participation

Their work though indicates abilities, which are esteemed among colleges and possible employers. The programmers list these projects on their resumes. This volunteer work can increase their lifetime revenue stream and reputation. It is feasible that they may have prospects for salary increase or promotion. From the perspective of software developer working on community open-source projects develops his non-firm specific knowledge, which increases his chances to be employed by another firm. Thus, if the programmer does not want to be tied to his current employer it is reasonable to invest time in open-source projects. That will also increase his bargaining power with his current employer. It is empirically verified that high-rank contributors (committers) in Apache Software Foundation projects earn higher compensations (I-H. Hann).

There is a study from Spence (1974) that shows the quality of workers by the level of education they pursue. Summarized it costs less effort to capable persons to pursue a degree. Thus, the less capable persons do not find worthwhile to pursue higher degrees, because the effort they made is not worth the potential gain in wage. The same principle could be an indicator for the quality and efficiency of the programmers, who contribute to open-source software.

5.2 Non-materialistic motives in open-source participation

The motivations may also be of non-monetary kind such as enjoyment or ego gratification (E. Haruvy, F. Wu, and S. Chakravarty). The self-fulfilment aspect of the work and success is decisive trigger behind the effort. The same applies to the need of recognition among the colleges. This work can result in fame and respect to the contributor.

Another motive of the developer could be the pure altruistic motive for the welfare of the society. That could derive from the perception of the developer that the large software companies possess too much power in unjustifiable way. This power should not be exclusive for the big corporation and the open-source projects are way to stand upon that. These convictions lead to increased contribution to the open source (E. Haruvy, F. Wu, and S. Chakravarty).

6 How similar/different are open-source and proprietary firms in their production and selling approaches?

At first sight we may think that the proprietary firms have better strategic planning and hence can better exploit market opportunities. The study of Campbell-Kelly and Garcia-Schwartz (2010) shows that the software development in proprietary and open-source firms link up in a similar way. Open-source firms invest on research and development and execute merger and acquisition of smaller firms to incorporate software development teams. They can manage their investments and portfolios as they acquire teams and software programs that others have developed. There are similarities in the production and the selling of software products when comparing proprietary and open-source software companies. To penetrate market opportunities, open-source firms also rely on dual-licensing strategies, but they restrict the access to the source code.

We will compare two different software projects. The first one has emerged as a proprietary product and the second one as an open source.

6.1 The evolution of Microsoft and its research and development costs

Microsoft was found in 1975 as a small company developing language translators. In 1980 they diversified in operating systems by licensing the source code of Unix from the owners of the company. Computers back were special machines and the idea of individuals owning a computer for personal needs was unusual. In 1980 IBM was searching for operating system to its personal computer. The Unix system was too heavy for the limited computational power of the personal computer. Microsoft presented an operating system to IBM, which one of the Microsoft's workers has hacked from Seattle Computer Products, replacing their name with Micro-soft (the initial name of Microsoft). They made a deal with IBM for the delivery of operating system code for their new personal computer. After the deal was made, they bought the operating system from Seattle Computer Products for 5000 US dollars. They improved it and shipped the product under the name MS-DOS. The first version of the software contained 4000 lines of code (Ichbiah and Knepper, 1991, p. 252). During the years it was updated and evolved. Interestingly Microsoft tried to migrate users to XENIX, which is operating system of Unix, but it was the market's choice to stay with MS-DOS.

Microsoft made three strategic decisions to solidify its position as a market leader. Back they had 40 percent of the market share. They developed a graphical interface for more user-friendly environment, invested 50 million in new generation OS/2 in cooperation with IBM (Zachary, 1994, p. 90) and created Microsoft NT, new multitasking software, which is a long-term investment for operating system powerful enough to replace minicomputers in enterprise computing environment (Schlender and Ballen, 1995). After a very successful launch of version 3.0 Microsoft gave up on OS/2 worsened the relationships with IBM. The plan was to make Windows the next generation operational system merging MS-DOS with NT. That required new investment in the NT project, and they followed. The new product had 4-5 million lines of code and included client and server components.

The 4.0 version of Windows NT came in 1996. It included new features like FrontPage website creation and management tool. Microsoft Explorer was developed for estimated 100 million dollars. There are not publicly stated information for research and development costs for Microsoft Windows, but the estimated cost is around 25000-30000 programmer years and can be calculated as 5-6 billion US dollars. This is in harmony with the stated from Microsoft research and development const of 28,5 billion dollars between 1989-2003, where big proportion of the costs are consumed in the development of NT (Campbell-Kelly and Garcia-Schwartz, 2010).

6.2 The evolution of Red Hat Linux its research and development costs

The software and computer coding has emerged as an open-source discipline in the 1950s. During the next decade, the proprietary firms imposed have imposed themselves on the market. It was not until early 1980s when the open source made a comeback due to Richard Stallman, who left behind the legal scheme for licensing, which ensured that derivative products from open source remain as such. The most popular open-source operating system today is Linux. It is developed by Linus Torvalds and other volunteer participants throughout the world. The first version of Linux is released in 1991 consisting of 10000 lines of code. In the next years with the volunteer work of developers the system progressed and became functional and robust. Version 2.6.0 released in 2003 contains 6 million lines of code.

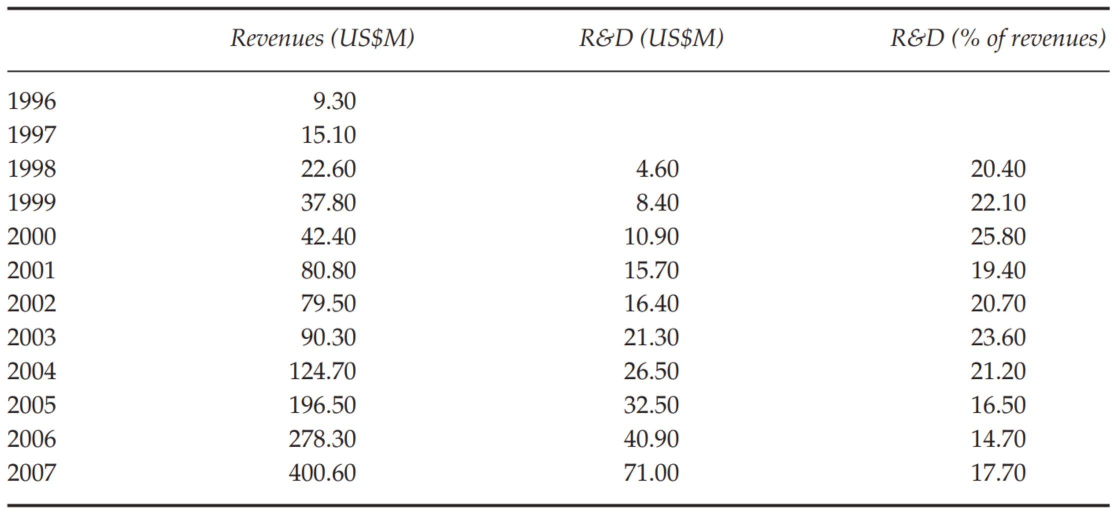
To create value for the user, much more than operating system is needed. What the user needs are applications like email, security programs, web servers, different kinds of business programs like database systems. These were created simultaneously with the progress of Linux. Their integration in the system and use though was complicated and not as easy as in proprietary software. Distribution system were developed from open-source software contributors, which were usually based in universities. They made the system more complete and ready to use, but software still lacked the needed support from the individual users and businesses. This gap was filled from firms, which developed business models for commercializing open source. Red Hat was one of these firms. They offered consultancy and assistance through telephone calls. By 1998 Red Hat were having 56 percent market share (Campbell-Kelly and Garcia-Schwartz, 2010).

Red Hat has two crucial contributions to Linux. The Anaconda Installer, which automates the installation, and the Red Hat Package Manager, which aggregates different elements into the software distribution. Because of the license agreement these contributions are again open source and other firms quickly copied the investment of Red Hat. But Red Hat had the strategy to release an updated version of the Red Hat Linux. The improved features of the new release, which is tested by a large group of users and the prestige gained by investing in research and development have paid off for Red Hat, whose software was awarded with product of the year in 1996 by Info World's (Campbell-Kelly and Garcia-Schwartz, 2010).

In 2001 the 7.1 version of Linux consisted of 30 million lines of code, which was close to Microsoft at that date. The estimated cost for development were 8000 developer-years, that is equivalent to 1 billion dollars. The expenditure for research and development of Red Hat for the first 5 years of the company were 40 million dollars. This amount is far from expenditure of Microsoft, but the expenditure's relation to revenues was the same as that of the proprietary giant.

The Assumption that the software products of open-source firms are dependent only on volunteer contribution is false. These firms have their own strategies and ongoing investments. The share of research and development spending to the revenues of the investigated case with Red Hat is larger than that of Microsoft and other proprietary firms. During 2001-2008 the open-source company spend on average 18 percent of their revenues on research and development. The share of Microsoft is 17 percent and those of IBM only 6 percent.

Here we can see the Red Hat's revenues and expenditure on research and development



	Revenues (US$M)	R&D (US$M)	R&D (% of revenues)
1996	9.30		
1997	15.10		
1998	22.60	4.60	20.40
1999	37.80	8.40	22.10
2000	42.40	10.90	25.80
2001	80.80	15.70	19.40
2002	79.50	16.40	20.70
2003	90.30	21.30	23.60
2004	124.70	26.50	21.20
2005	196.50	32.50	16.50
2006	278.30	40.90	14.70
2007	400.60	71.00	17.70

Image source (Campbell-Kelly and Garcia-Schwartz, 2010).

6.3 Similarities between business approaches between open-source and proprietary firms

As the microprocessors became more powerful and their costs sunk, the client-servers became more affordable in the 1990s. The microprocessor producer Intel was effective enough to serve to servers. This significantly reduced the price for businesses to use the new technology. The operating systems of Microsoft and Linux were more suitable for these new generation server processors because they were cheaper than the established Unix.

6.3.1 Red Hat's acquisition costs

In this section we want to show that there is a convergence in the software industry concerning the strategies of the open source and proprietary firms. The assumption that the open-source firms rely only on volunteer programmers to develop its products is false, as discussed in section above. To develop a winning business strategy the open-source firms, have variety of different costs such as for acquiring other firms' products and competent personal. Red Hat is again a good example for showing business strategy case for needed acquisitions for prosperity in the market and interrelations with other firms, both open-source and proprietary.

We take for usual that proprietary firms do acquisitions to leverage their products. Oracle made 56 acquisitions in the years between 2001 and 2008. The value of the acquired companies was 34.2 billion dollars. During the same period, the top 10 software firms spend 100 billion dollars. But they were not only proprietary firms that engaged in these strategies.

In 1999 the initial public offering of Red Hat took place. With the raised capital the company had the vision to offer a full e-commerce solution. There were different companies specialized in different fields such as payment processing (Hells' Kitchen Systems), application development software (Cygnus) and website security (C2Net). There were also other firms, which were acquired from Red Hat (Planning Technologies, Aktopia, Bluecurve). The common between these firms was, that they were all open source and Red Hat could use their products as free rider. But the acquisition of the firms puted Red Hat in a suitable position for their consulting services. Another tale of the story is the acquired talent from these companies. As we see in this example the behaviour of Red Hat was no different than other proprietary firms. They had research and development costs for the development of their services and acquisition costs. We can state that to have a stable position in the market, open-source firms have similar cost structure (when it comes to research and development) as the proprietary firms. Free riding is not the proper strategy for long term, scalable and large investments.

6.3.2 Transition to dual-licensing of open-source firm

The business strategy of Red Hat was very promising, and their revenues grow constantly. They offered the Red Hat Linux software bundled with support service to individual customers and professional service to businesses. IBM, Compaq, Dell, and HP implemented the operation system in their devices, because the customers were interested in products with support services. That gained Red Hat a stable market share. Another positive circumstance was that proprietary vendors like SAP and Oracle certified their products only for primary distributions firms (mainly Red Hat and Suse).

Even though Red Hat succeed in establishing on the market, their strategy still had a crucial pitfall. Because of the GPL license, their product was available on the internet and other firms could duplicate it and sell the product through price-dumping. As a solution Red Hat decided to go for a change in their strategy and they implemented the dual-licencing strategy. Red Hat Linux was abandoned, and the Red Hat Enterprise Linux (RHEL) was introduced together with Fedora, which was the free version that ensured the further development of the Linux from the volunteer community. RHEL was stable, commercial product for businesses. The product was bundled with support and maintenance. It also included seven years guarantee and upgrades from the Red Hat's infrastructure.

The dual-licensing structure combined two advantages. First, Red Hat could profit from the development coming from the open-source community to improve their product. Second, the copyright cum license enables the company to make revenues. This can be viewed as price discrimination, as the customers who do not want to spend time compiling the code can buy the RHEL subscription.

The strategy of dual subscription increased the share of the revenues made by software licenses. At its founding years the company counted on services as sources of income. At the end 1990s the half of the company's revenues came from services, whereas in the last 3 years until 2008 the share of the subscriptions has grown in a such way that they represent more that two thirds of the incomes.

Interestingly the opposite trend was true for propritary firms. While open-source firms gained new shares in subscribtions, the proprietary firms increased their revenues in services. We observe convergence between open-source and proprietary enterprises.

6.4 Applications layer production and selling strategies

The application software is oriented towards versatile needs and there can be found very profitable niches. Thus, it can be stated that the applications software market is a place of various opportunities in comparison to more fixed software layer.

While there are open-source software products for operating systems and middleware, open-source applications for some sectors like finance. One reason for that is, that writing application requires to shape them in a user-friendly way. This increases the cost of the development and requires user training programs and usability tests (Campbell-Kelly and Garcia-Schwartz, 2010). Successful application can be written when the developers can see through the eyes of the end-user. One example for successful software is the Firefox browser. While browsing in internet the experience for developer is not different as to the normal user.

Another obstacle for the applications is the need for specific domain knowledge to frame them. To write applications for industries like finance and healthcare the designer of the program must have sophisticated understanding of the requirements of these services. In the 1990s when the open-source community was dominated by young programmers, who have newly graduated from universities, the young developers did not have the needed expertise and knowledge to develop this kind of projects. With the maturation of the open-source paradigm this is turning around and 'innovation networks' emerge as called from von Hippel (von Hippel, 1988).

6.4.1 Development of proprietary firm SAP

During the late 1960s the development of mainframe application software was expensive and project we usually based on contractual orders specific for a given domain. At some point it become reasonable to offer a service as software package instead of custom solution. This made the products more affordable for the customers and development costs were reduced. SAP is found in Germany 1972. They wrote a custom-written accounting package for enterprise in chemical industry and kept the rights for the software. They enlarged the basis of customers and industries they work with and accumulated great amount of knowledge. In 1981 they launched R/2 ERP. This was the transition from service to multifunctional product.

The constant evolution of SAP occurred thanks to the accumulation of knowledge and capabilities in business processes and organizational domains. In addition, SAP would augment its software when that was requested from the customer to adapt to its need. In an open-source version the customer would supply the code for the new future, in SAP case customers requested the firm, that would do that for them and keep the ownership over the code.

6.4.2 Development of open-source firm Compiere

Compiere is found in 1999 and its early years targeted small and medium firms for its ERP software. Proprietary firms did not target the small firms because of the high costs and complexity of the software. Compiere's product was made of basic modules in inventory management, accounting, and customer relationship (Campbell-Kelly and Garcia-Schwartz, 2010). The sources of income were as usual for open-source firms the support services and training offer.

For the development of their product, they used third party consultant for the framing of the software-capabilities. The free access to the software made possible the direct implementation of the new features from third parties, which reduced costs by the development. A proprietary firm would request the new feature and write the code themselves. To remove obstacles for derivative works Compiere used Mozilla Public License. That would allow the customer to not share their modification to keep competitive advantage and confidentiality. Nevertheless user-organizations made huge impact on the development. For example, one user shared credit-card processing module (Koch, 2004). This allowed the company to have user-driven innovation, which was discussed in section 2, subsection "How open source develops your product"as an advantage of open source.

Both Compiere and SAP created communities, which helped to innovate their products. The difference was that in the case of SAP the contributors and consultants identified the needed functionalities, whereas by Compiere they could write into the source code either and therefore we can state that they brought it to the next dimension and reduced innovation costs alongside.

6.5 Equalization in the selling approach

As we discussed already, SAP relies on both license services and consulting services for generating its revenues. Compiere's business strategy relied fully on support and training services. That, though, changed, when Compiere brought the three different versions of its product to the market. Community edition with GPL license, that can be freely downloaded. Standard edition with GPL license, that includes support and enhanced features. Finally, the professional edition with MPL license, that is the most advanced version and free from the engagements of GPL license. Here we see that there is a convergence in the production and selling approach between open-source and proprietary companies. Both Compiere and Red Hat rely on dual-licensing model to generate values. On the other hand, proprietary firms also give free versions on disposal for reaching the needed large scale of user size and to reduce support costs.

7 Conclusion

The existence of open source changes the structure of the market and redefines the ways the firms organize their business strategies. In this seminal work we suggested that every software development project is an investment activity that has cost-benefit structure and suggested that the implementation of the product by different engineering methods must be with coherence of the firm's long-term strategy. While managing the product portfolio, the firms should consider the future possibilities for synergies and opportunities that may arise. By defining product portfolio tactical matters should also be concerned, as better models for cost estimation or the effect of learning curve.

We examined what kinds of costs come with software development projects and showed the benefits that a free version may provide to reduce these costs. Finally, the examination of business case studies with open-source and proprietary firms, showed that there is a trend for convergence in the way software is produced and sold. Both open-source and proprietary firms have research and development costs, that represent similar shares of their revenues. They both execute acquisition strategies, as they buy synergetic and complement products from other companies. Historically open-source companies relied on consulting services, now they sell products using dual-license strategy. Both types of firms use the advantages of open source and the contrasts become less significant.

8 Sources

Boehm, Barry W.; Sullivan, Kevin J. (1999) Software Economics (University of Southern California and University of Virginia).

Campbell-Kelly, M.; Garcia-Swartz, D. (2010) The Move to the Middle: Convergence of the Open-Source and Proprietary Software Industries, International Journal of the Economics of Business, 17:2, 223-252 Collins, J.C.; Porras, J.I. (1997) Built to Last: Successful Habits of Visionary Companies (Harper Business).

Cusumano, Michael A. (2004) The Business of Software (New York: The Free Press).

Haimes, Y.Y. (1998) Risk Modeling, Assessment, and Management (Wiley).

Ichbiah, D.; Knepper, S.L. (1991) The Making of Microsoft (Rocklin, CA: Prima Publishing).

Koch, C. (2004) Open-Source ERP Gains Users, CIO, 1 February.

MIT (2008) An interview with Marten Mickos: the Oh-So-Practical magic of open-source innovation. (MIT Sloan Manage Rev 50(1), 15).

Riehle, D. (2007) The economic motivation of open source: stakeholder perspectives (IEEE Comput, 40(4):25–32).

Riehle, D. (2012) The single-vendor commercial open course business model (Inf Syst E-Bus Manage 10: 5–17).

Schlender, B.; Ballen, K. (1995) What Bill Gates Really Wants, Fortune, 16 January. Shankland, S. (2006) Oracle has yet to prove Linux cred, CNET News, October 27. Avail

Statista (2020)`https://www.statista.com/statistics/203428/total-enterprise-software-revenue-forecast/#:~:text=Enterprise%20software%20total%20worldwide%2`
`&text=In%202021%2C%20IT%20spending%20on,percent%20from%20the%20previous%20year`

Trigeorgis, L. (1997) Real Options: Managerial Flexibility and Strategy in Resource Allocation (Cambridge, Massachusetts: MIT Press).

Zachary, G.P. (1994) Showstopper! The Breakneck Race to Create Windows NT and the Next Generation at Microsoft (New York: The Free Press).

von Hippel, E. (1988) The Sources of Innovation (New York: Oxford University Press). von Hippel, E. (2005) Democratizing innovation (MIT Press, Cambridge).