Influence of release frequency in software development

Alexis Benoist

Influence of release frequency in software development

Alexis BENOIST
Computer Science department
KAIST
Daejeon, South Korea
alexis.benoist@gmail.com

***Abstract*—**

The length of the development cycle is an extremely important parameter in software engineering. It's a crucial variable because it's the one which will determine how often the user will have the updates and decide how the development process should be carried on, which means how the developers will work. The features have to be incorporated as soon as possible to deliver the maximum value to the user, though the program must be tested enough to be stable. So the release frequency is a complicated variable to adjust. First we will give a model to quantify the quality at different release rhythm. Then we will use that model to optimize software development in term of size and profit. Finally, we are going to discuss the advantages and drawbacks of different development length on the qualitative side.

Keywords—software engineering, release frequency, economy

I. INTRODUCTION

A. *Importance of release frequency:*

Most organizations which produce software are actively looking for the ideal development length for a release. Google started by changing the development frequency of Chrome from a release every 12 weeks to one every six weeks [1]. Mozilla followed by also giving Firefox also a six weeks development lifecycle[2]. It's also the case for exploitations systems. Microsoft wants to release a new version of Windows annually [3], which is a real increase of the release frequency, the past versions where released between every other year and a 6 year gap between versions. On the other hand, Canonical may change the release development for a 6 months lifecycle to a 2 years release frequency [4].

B. *Technical and business influence of SaaS software:*

Also, Software as a Service (SaaS) allowsmaking updates really easily as they only have to update the server to touch every user. As the cost to release a new version is marginal, the software published should optimize this time to get maximum profit and customer satisfaction by introducing new features as soon as possible but avoiding bugs. It the traditional software development market, the editors waited to have significant improvement over the old version to release a new one. On the opposite, in the SaaS or software renting market they want to release the new features as soon as possible to improve the attractiveness of the product over the completion. Software is going from perpetual licensing to renting. We can give the example of Microsoft Office[6], even though it's traditional software ithas the possibility of being, and it will be more and more rented as Kurt DelBene, president of Microsoft's Office division says "I would say in 10 years, the majority of customers, perhaps all customers, will be in a subscription relationship as opposed to a perpetual"[6].Also, the world of business thinks in this way as we see in the study "Delivering software as a service" in The McKinksey Quarterly [8].In this paper we will focus on the renting/SaaS business model.

II. PRESENTATION OF THE HYPOTHESIS AND THE MODEL

A. *Model of code production:*

1) Comparison with previous research:

We will follow the step of Vidyanand Choudhary's, by keeping his main hypotheses.So he modelsthe releasing process as a linear improvement of quality over time [5]. Even though the release cycle is very frequent, it's not continuous, so we will model it with a step function. We will try to model differently two main things, which are:

- The quality of the service over time will not be a linear function but a step function.
- The quality of the software will not grow at the same pace all the time but we will suppose that the investments are constant

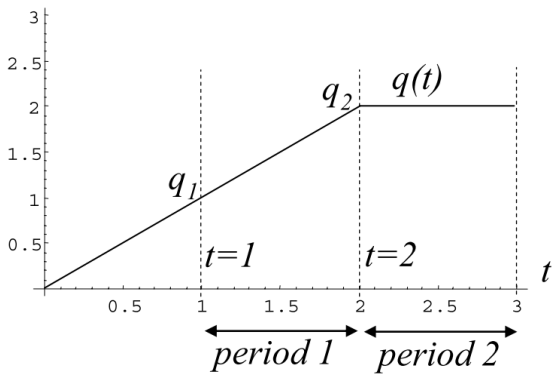

Figure 1 – Representation of the quality over time of SaaS software with Vidyanand Choudhary's model

We can see the illustration from the model from the literature and the one we are exposing here in figure 1 and figure 2. First of all, the software release even though they are really frequent they are not continuous so they should be modeled as a step function. Then, the cost of the software can be found using the COCOMO II model.

Also if the software grow at the same pace, the investments will have to increase because it needs more efforts to modify a software of a bigger size. We will suppose that the company has a R&D department of constant size, so invest the same amount of money.

These new assumptions give us a new parameter that we will mainly study: the influence of the release frequency which we will call T. There will be new release at each time multiple of T. So at each time T, there will be a release of the new features and there will be an augmentation of the software quality.

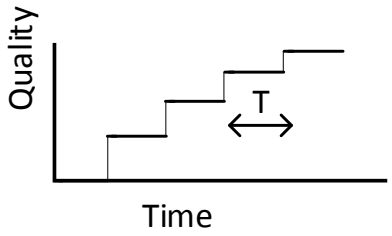

Figure 2: Illustration of quality over time

2) Hypotheses:

The quality q is a step function as we see in figure 2. The quality q is defined in equation (1) as proportional to the number of Source Line Of Code (SLOC) released.

$$q(t) = a \times SLOC_{released}\ (t)\ (1)$$

We need to define how we will model the production of the software. For that we use the equation (2) given by the COCOMO II manual model [7].The cost of the program of a determined Size is proportional to the monthly nominal salary and to the size of the program power E.We are going to explain the different parameters.

A is the effort coefficient currently set to 2.94, but it can be calibrated.

PM_{salary} is the nominal monthly salary for a developer.

$Size_{developed}$ is the size of software developed for that cost.

E represents the economies (if less than one) and diseconomies (if greater than one) of scale.

The Effort Multipliers (EM_i) are model to adjust the nominal effort to reflect the software product under development. They take onto account the specificities of the product, the platform, the personnel and the project. The important thing to know is that if the organization is totally nominal, we will have $\prod EM_i = 1$. Also if the organization is not efficient in some ways or if the products have some complexity, the effort multipliers will be multiply the cost by a value which is superior to 1. On the other side, if by instance the people have some experience in the domain, the effort are going to be less than nominal. We won't describe them into details, to know more about them, you can read the COCOMO User Manual [7] in which they are explained in detail. We will consider that the $\prod EM_i$ is a parameter of the project.

$$c = A \times PM_{salary} \times Size_{developed}^{E} \times \prod EM_i (2).$$

Then size of the program is, as defined in COCOMO II manual model [7], described in equation (3).

$$Size_{developed} = NewKSLOC + EquivalentKSLOC (3)$$

We model the equivalent Thousand SLOC (KSLOC), so the number of KSLOC in legacy code which needs to be modified as a linear function of the number of new KSLOC developed and of the number of KSLOC in legacy code, as we define in equation (4). As the legacy code is more complex, we need to modify more of the existing program.

$$\text{Equivalent KSLOC} = \text{AAM} \times \text{adaptated KSLOC} (4)$$

The Adaptation Adjustment Modifier (AAM) is a coefficient defined in COCOMO II, to assess the ratio of code which has to be modified. The adapted code is the code from the previous release which will be modified to be used in the next version of the software. For more information about this parameter, you can refer to [7].

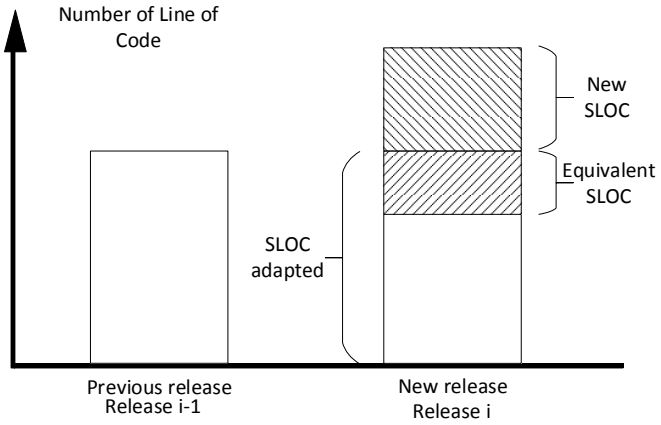

Figure 3: Representation of the size of two successive releases

To help to make the difference about the used terminology we explain all the terms in figure 3. The striped area represents the Size of the program which has to be developed during this iteration.

We want to emphasize the fact that $Size_{developed}$ and $Size_{release\ i}$ are different. $Size_{developed}$ is the size developed during a development cycle (striped area). $Size_{release\ i}$ is the size of the release, which is the total height of the bar.

We chose the cost c for a period of time T to be equal to the budget b allocated per unit of time, as we see in equation (5).

$$c = b \times T \quad (5)$$

The number of KSLOC released at the end of the period i, is computed described in equations (6) and (7) which is a combination of the equations discussed previously.

$$Size_{release\ i} = New\ KSLOC + Size_{released\ i-1} \quad (6)$$

$$\text{Size}_{\text{developed i}} = \text{New KSLOC} + \text{AAM} \times \text{Size}_{\text{released i}-1} \quad (7)$$

We assume that the number of reused KSLOC is the size of the previous release, which means that all the existing features are kept in the next version.

With that we can express the equation for the number of new KSLOC (8).

$$New\ KSLOC = \left(\sqrt[E]{\frac{b \times T}{A \times PM_{salary} \times \prod EM_i}} - AAM \times Size_{release\ i-1} \right) \quad (8)$$

Also we can express the number of KSLOC in the release I as the sequence (9):

$$Size_{released\ i} = \sqrt[E]{\frac{b \times T}{A \times PM_{salary} \times \prod EM_i}} + (1 - AAM) Size_{release\ i-1} \quad (9)$$

Of course, the size of the release should be strictly superior to the previous release. As we just model the quality in function of the SLOC and we will not go over code refactoring or changes in architectures. Even though, depending on the size of the previous release and the different parameters, the size of the release can decrease, in this case we will just consider it constant. Excluding this possibility we can express the size of the release in function of its iteration number, as described in equation (10) and (11), if:

- AAM is different from one (which means that we use some of the code from the previous release)
- all the parameters are constants

$$Size_{released\ i} = (1 - AAM)^i (Size_{release\ 0} - r) + r \quad (10)$$

$$r = \frac{\sqrt[E]{\frac{b \times T}{A \times PM_{salary} \times \prod EM_i}}}{AAM} \quad (11)$$

Because there is no software at the initial state, we can assume that $Size_{release\ 0} = 0$, so we get the simplified equation (12):

$$Size_{released\ i} = \frac{\sqrt[E]{\frac{b \times T}{A \times PM_{salary} \times \prod EM_i}}}{AAM} \times [1 - (1 - AAM)^i] \quad (12)$$

B. Analisys of the sequence of the size of the software $Size_{release}$ *:*

Some parameters have an influence on the scale only of the sequence and some others influence the behavior of the sequence as summarized in Table 1.

Influence the scale only	Influence the behavior
b	E
T	AAM
A	
PM_{salary}	
EM_i	

Table1 - Nature of the coefficient of the size sequence

The analysis of the sequence modeling the size of the software is very important, because the size of the software can be an objective which would influence the choices of the parameters as the size of the team (which is visible through the ratio: b/PM_{salary}) and the length of the increments. The objective may be to create a software of a certain size in a decided schedule.

The sequence is bounded if AAM is non-zero.Mathematically, it's explained by the factor AAM,which models that the size developed has a part proportional to the size of the previous release. Once the size of the program is near its limit the software engineers spend their time understanding and programming again what was in the

previous release. It seems realistic because he size (so the complexity) of the software cannot go above the capacity of humans of handling complexity, which is bounded.

If AAM is zero, the size of the software is not bounded. As there is no rework on the previous developed code the software is growing at a linear pace, which means that at each iteration the quantity of new code is the same as we see in equation (13).

$$Size_{released\ \ i} = i \times \sqrt[E]{\frac{b \times T}{A \times PM_{salary} \times \prod EM_i}} (13)$$

We expose the different possible limits in equation (14).

$$if\ \alpha = 0, \qquad \lim_{i \to \infty} Size_{released\ \ i} = +\infty$$

$$else, \qquad \lim_{i \to \infty} Size_{released\ \ i} = \frac{1}{AAM\sqrt[E]{A}} \times \sqrt[E]{\frac{b \times T}{PM_{salary} \times \prod EM_i}} \quad (14)$$

So the final size of the software is inverse proportional to the factor AAM and proportional to the root of E (factor of economies or diseconomies of scale) and $\prod EM_i$(product of the effort multipliers)to:

- The number of people involved in the project
- The length of a development cycle

So to increase the maximum size of the software we have to increase the intelligence working on it. Because the human capabilities of understanding in a limited time are limited.

Now that we have looked at the final size of the software we can look at its evolution over time.

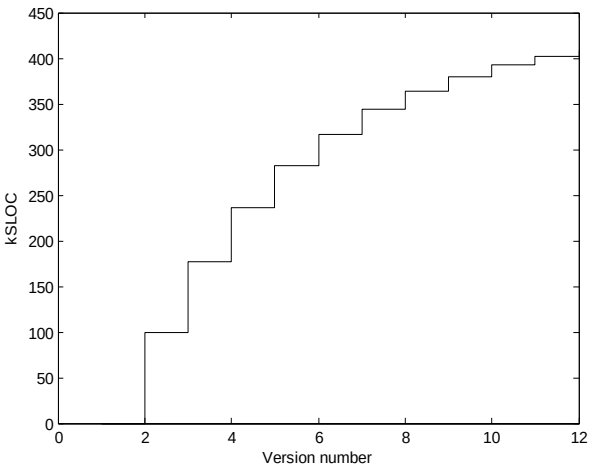

Figure 4: Modeling of the size released of the software in function of time

We can see in figure 4 that the number of SLOC added at each iteration is decreasing, because the complexity of the software is increasing and the developers dedicate more time to rework and understanding legacy code.

III. QUANTITATIVE ANALYSIS

A. *How to maximize the size of sofware given a certain schedule and budget:*

1) Hypotheses:

The objective of a firm can be given certain time and resources to create the best software as possible, which with we model as maximizing the size of the software at a given time t_{maxize} , which we define in equation (15) with i the number of iterations.

$$t_{maxize} = i \times T (15)$$

Given this assumption, we can write the formula in equation (16).

$$Size_{\substack{released\ \ at \\ time\ t_{max} \\ i}} = \frac{\sqrt[E]{\frac{b \times t_{maxize}}{A \times PM_{salary} \times \prod EM_i \times i}}}{AAM} \times [1 - (1 - AAM)^i] (16)$$

By convenience, we will call $Size_{\substack{released\ \ at \\ time\ t_{max} \\ i}}$, $Size_i$.

2) Comments on the formal method:

$Size_i$ is a sequence, to know its optimum we will compute the ratio $Size_{n+1}/Size_n$ in equation (17).

$$\frac{Size_{n+1}}{Size_n} = \sqrt[E]{\frac{n}{n+1}}\left[\frac{1 - f^{n+1}}{1 - f^n}\right] = \sqrt[E]{\frac{n}{n+1}}\frac{\sum_{i=0}^{n} f^n}{\sum_{i=0}^{n-1} f^n}$$
$$\text{With} f = (1 - AAM) (17)$$

The multiplier $\sqrt[E]{\left(\frac{n}{n+1}\right)}$ is smaller than 1 whereas the multiplicand $\frac{\sum_{i=0}^{n} f^n}{\sum_{i=0}^{n-1} f^n}$ is bigger than 1, the criteria to know the variation of the series is $\frac{Size_{n+1}}{Size_n} > 1$ or $\frac{Size_{n+1}}{Size_n} < 1$ we cannot formally determine the comportment of the series.

3) Simulation based method:

We can see that the ratio of equation (17) is only function of the number of iteration n, the economies of scale E and the AAM parameter, so giving these parameters we can determine the best number of iteration to do to have the best software. So an organization can determine the optimal number of versions to do using simulations.

We could study the Size in function of the continuous parameter n, to find the optimal number of versions in a formal way but there wouldn't be a lot of added value because the number of versions is intrinsically discrete.

To find the results we compute the formula for all the values of i smaller than a certain number and we simply look for the biggest Size.

4) Results:

By analyzing the function we see that when the factor of efficiency is smaller than 1, the best number of versions to do is only one.

If there is efficiencies in the productivity function, which means that $E > 1$, the number of version is not always one, as we can see in the figure 5.

We will compute the optimal number of versions for different parameters.

		AAM								
		0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
E	1,1	2	1	1	1	1	1	1	1	1
	1,2	3	2	1	1	1	1	1	1	1
	1,3	5	2	1	1	1	1	1	1	1
	1,4	6	3	2	1	1	1	1	1	1
	1,5	7	3	2	2	1	1	1	1	1
	1,6	8	4	2	2	1	1	1	1	1
	1,7	9	4	3	2	1	1	1	1	1
	1,8	10	5	3	2	2	1	1	1	1
	1,9	11	5	3	2	2	1	1	1	1

Figure 5 - Optimal number of versions to develop given AAM and E parameters

We can see that the effort multipliers don't affect the number of versions to do, as it only change the scale of the function and not its behavior.

The more AAM factor decreases the more versions we need to optimize the size of the software. Also, when the efficiencies increase the more versions are needed to optimize the size of the software.

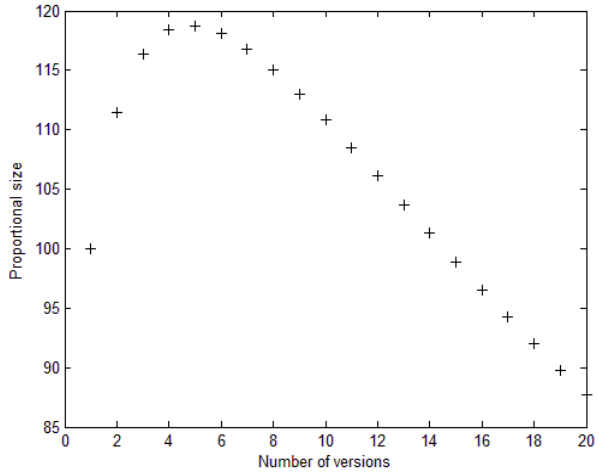

Figure 6- Normalized size in function of the version number
(with coheficients E=1,3 and AAM=0,1)

The figure6 illustrates how the model behaves. We can see very clearly that the maximum size is when there are 5 versions, as we also see in the figure 5. We normalized the size to have the first one be equal to 100, so we can see the size difference as percentage.

We can also see that the size difference between doing only one version and 5 versions, make the difference of size of 18,7%, which is not negligible at all.

B. Economic analisys:

1) Problem:

The problem we will try to solve is answering the question: "What is the optimal number of versions given budget and schedule to optimize the company profits?"

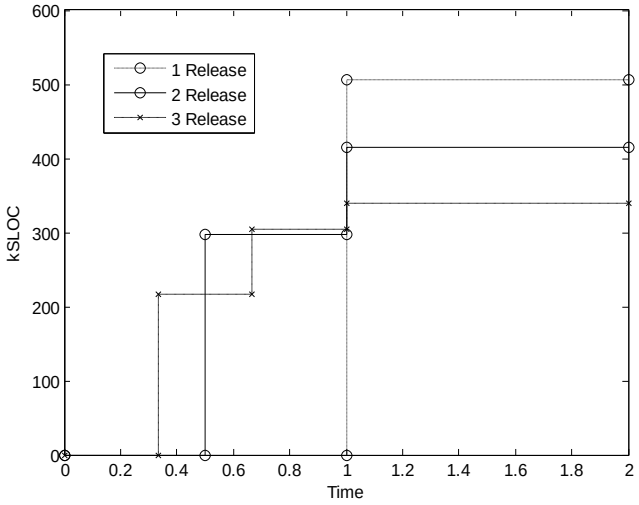

Figure 7 - Size of the versions over time with different number of releases
(E=1.3 and AAM=0.6)

In figure 7, we can see the size of the version with given resources but with different number of versions. We can wonder what is the best alternative for businesses starting doing business early or to have a better quality at the end of the development.

2) Model:

This model will allow us to study the influence of the release frequency on the quality of the software released given E and the AAM parameters. We can study this phenomenon in the case of equal and constant investment.

We will use the economic utility function u to determine the willingness of the buyer to pay for the product. We will model the heterogeneous taste of the customers by the θ parameter, which is supposed to be uniformly distributed in [0, 1] as we can see in equation (18).

$$u(t) = \theta \times q(t) \quad (18)$$

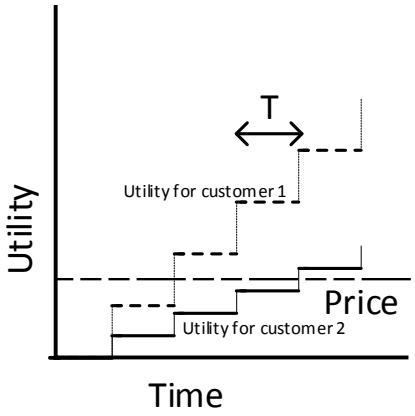

Figure 8 - Representation of utility and price

In the figure above we can see the representation of utility for two different customers of the quality function of figure 2. Customers only rent the product when the utility is higher than the price. We can see that the customer represented by the full line will only buy the last version. The other customer will rent the product only from the 2nd version and will continue after. Once a customer rented the product once, he will continue to rent it because the quality, so the utility is increasing over time.

T is the release period, but it will be the billing period too to simplify the calculations.

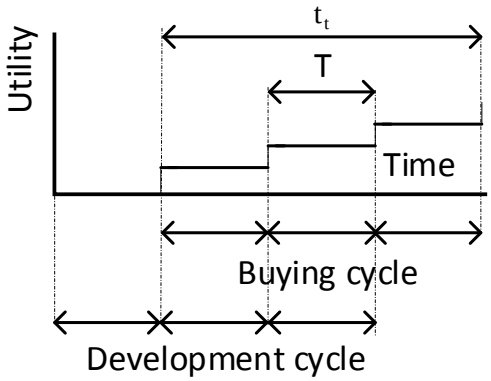

Figure 9 - Illustration of the buying and development cycles ($x_{max} = 3$)

As we see in figure 9, there is the same number of development and buying cycles and they have the same period T. Although, they are not at the same time. We consider that the company stop to invest in the last buying cycle because it won't have any utility in the model, but in reality the development would continue. There is a development cycle before the first buying period to develop the first version and not possibility to buy the software during that time because no version has been released yet.

We assume that the customer rented the software at a rent p per period of time$T$, which is the rent of the product per unit of time, which means that the product $p \times T$ is constant.

We expose in figure 10 the process to find the profit. We first of all need to find the price from the parameters. Then we look for the least interested buying customer for version i, θ_i and finally we can find the profit.

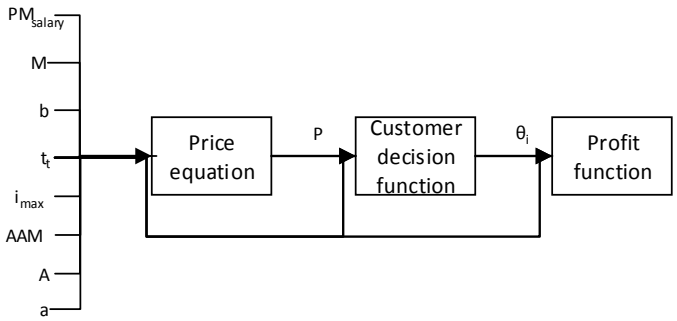

Figure 10 - Process to find the profit

We have to compute the surplus of the consumer during a time $t_t = x \times T$ which is a multiple of the period of release. We assume that the customer will start to use the service at $t_0 = x_0 \times T$. In order to do that we have to introduce the price, which we will call p.

$$s(t) = \int_{t_0}^{t} (u(\tau) - p) d\tau \quad (19)$$

As the function is a step function, we can integrate easily (19) to get the equation (20). We use the definition of the utility in equation (18), then we use the equation (1) to express the quality. As all the parameters are constant we can transform the continuous integral into a summation. We consider

$$s(t) = T\left[\theta a^{E}\sqrt{\frac{b \times T}{A \times PM_{salary} \times \prod EM_i}} \sum_{i=x_0}^{x} \frac{[1-(1-AAM)^i]}{AAM} - (x - x_0)p\right] \quad (20)$$

θ_i is the least interested customer buying the product who will rent the product at the release i, as θ is homogeneously defined, the proportion of the population M, who will rent the version i is $(1 - \theta_i)$. We can find θ_i by solving the equation $u(\tau) - p = 0$ at the time τ of the release of the version i. We can see the expression of the least interested customerbuying the product in equation (21).

$$\theta_i = \frac{p}{a} \frac{AAM}{[1-(1-AAM)^i]} \sqrt[E]{\frac{A \times PM_{salary}}{b \times T \times \prod EM_i}} \quad (21)$$

We consider the profit made by the company in equation (22).

$$\pi = \sum_{i=1}^{i_{max}} TpM(1 - \theta_i) - b\, t_t \quad (22)$$

We can present it as in equation (23):

$$\pi = TpM(i_{max} - \sum_{i=0}^{i_{max}} \theta_i) - b\, t_t \quad (23)$$

We can easily derive the profit π in function of the price p, to find the optimum in equation (24).

$$p = \frac{a\, i_{max}}{2\sum_1^{i_{max}} \frac{AAM}{[1-(1-AAM)^i]}} \sqrt[E]{\frac{A \times \prod EM_i \times PM_{salary}}{b \times T}} \quad (24)$$

Now we have expressed the profit as a function of AAM,i_{max} ,t_t, b and M. We will focus on the influence of i_{max} , the number of versions released.

3) Results:

We will study the influence of the different parameters. We will use on the numerical simulations the following assumptions. The effort multipliers are nominal. Only one person is working. The length of the project is 12 months. The utility is proportional to the quality by a factor 100.

a) Influence of the schedule:

We are going first to see the influence of the schedule on the simulation. We can see these results on the figure 11.

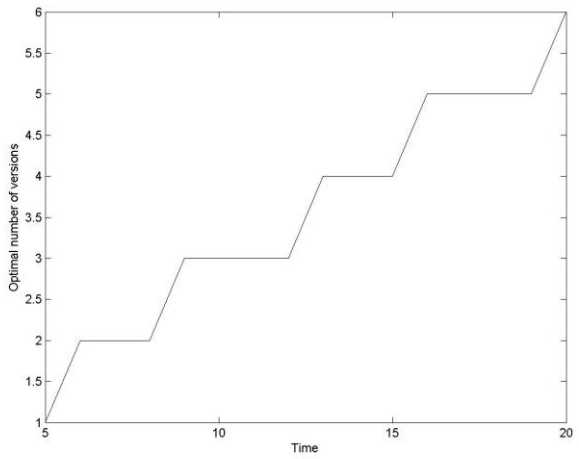

Figure 11 - Influence of the time allocated on the optimal number of releases

We can see that quite logically the number of versions to do is increasing with the time available. The shape of the curve is due to the fact that we can only do an integer number of versions.

b) Influence of ratio between the utility and the software a

We made vary this ratio between 1 and 100 and the optimal number of versions is 3 for all this range of values. We consider that this parameter doesn't change the optimal number of versions to do.

c) Influence of the size of the market

We simulate for a market from size 1 to 1000 and it didn't changed the results of the simulation, the optimal number of versions being three versions.

d) Inflence of the number of developers

The influence on the number of developers is described in the figure below.

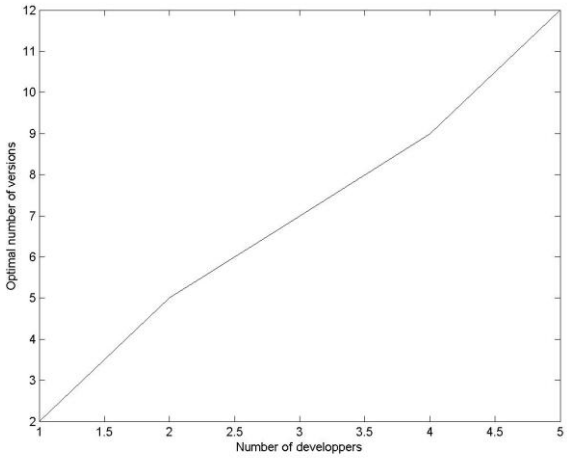

Figure 12- Influence of the team size allocated on the optimal number of releases

We can see that the more people there is, the more versions you need to do. The model considers that the working capacity is better if there is more people, so there should be more versions. What we don't consider is if the size of the team increase the number of internal communications will increase as the square as the number of people. This could be modeled by a decrease of the E parameter.

By looking at the formulas, we can say that what matters is the ratio $PM_{salary}\ /b$ which is the number of developers.

e) Influence of the efficiency E and the rework factor AAM

We can see on the figure below the optimal number of versions to develop in function of AAM and E parameters.

		AAM								
		0,1	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**
E	**1,1**	4	4	4	4	4	3	3	3	3
	1,2	5	4	4	4	4	3	3	3	3
	1,3	5	4	4	4	4	3	3	3	3
	1,4	5	4	4	4	4	3	3	3	3
	1,5	5	4	4	4	4	3	3	3	3
	1,6	5	5	4	4	4	3	3	3	3
	1,7	5	5	4	4	4	3	3	3	3
	1,8	6	5	4	4	4	3	3	3	3
	1,9	6	5	4	4	4	3	3	3	3

Figure 13 -Optimal number of versions to develop given AAM and E parameters

We can see that when the AAM factor is higher, the optimal number of versions is decreasing. We can explain that by the simple fact that if AAM is bigger, the amount of rework

to do is higher. In that case the developers should do fewer versions do less rework.

If the efficiency is higher, the number of versions to do is higher. We can explain it because the effort that the developers can do is higher when the efficiency is higher, in consequence they should do more versions.

We also see that in this simulation that the AAM factor (1 to 3 version in difference) influences more the optimal number of versions than the efficiency E (only one version of difference).

f) Influence of the Effort multipliers:

We are going to see the influence of the product of the effort multiplier on the simulation. We can see these results on the figure 14.

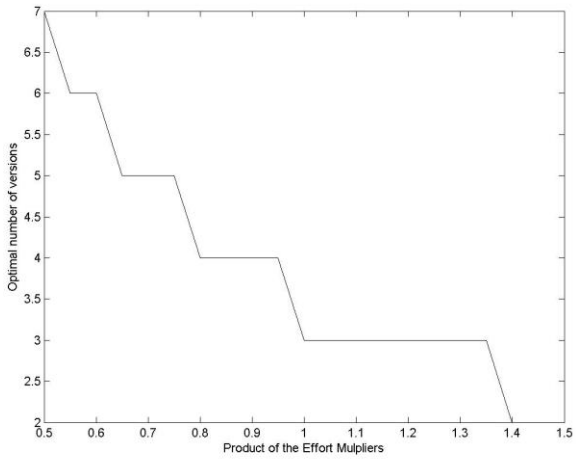

Figure 14 - Influence of the product of the effort multipliers on the optimal number of releases

We can see that the number of versions to do is decreasing when the product of the effort multiplier increases. That means that the more the team is skilled, efficient and has domain specific knowledge the less versions they have to do. It can be related to the fact that if a team is skilled it needs less iterations to do the right product.

IV. ANALYSIS OF NON QUANTIFIABLE PHENOMENONS

In this part, we will discuss the advantages and drawbacks of different release frequency in function of their attributes, such as the size of the software and its business model. We will discuss software release frequency in a general software sense and not only in the SaaS and renting software.

A. Influence on the customers:

A high frequency release, means that the new features will be available for the customer. For personal software, it's most of the time a good thing and it can be a competitive advantage to have new features often and not to have outdated software for some years. But in the business world, updates can be a problem because they can disrupt the habits of the users. To counter that making changes gradually as Apple is doing to make closer its computer operating system and its mobile one [15] , on the opposite of what Microsoft did with the start screen of Windows 8 [14] and the ribbon of Microsoft Office 2007 [13].

One of the problems is the difficulty for the customer to keep up with updates. If the release frequency is too high there are two reasons why the customer may not want to upgrade: the updating process is complicated or costly. Another possibility for the customer not to want to do the upgrades may be the lack of perceived added value for the customer.

The main problem may be for business applications such corporate information system. The IT direction may use a combination of several Commercial of The Self (COTS) products. The updates may change the behavior or the interfaces of the product. The IT direction may be willing to use the software as long as possible not to have to test the new system or to have to redevelop glue code between the components to make them talk to each other. In this case the stability of the functionalities should be important but how can a software company make sure of the stability of the program.

Also the release frequency should not increase the presence of bugs and the developers should develop the right processes not to decrease software quality and reliability by the introduction of bugs which have a really bad influence on the customer view on the product.

B. Influence on the working conditions:

1) Parkinson's law

Giving a time and members to a project makes is giving it resources. In [9], Mehwish Nasir explainsParkinson's Law in software projects as "the project costs whatever resources are available and the project's life cycle is expanded as per the number of resources available in the organization."

In Parkinson's view most of the efforts to achieves objectives are made just before the deadline, because the incentive for the people is not to miss the deadline and most people have troubles feeling the deadline if it's in a long time. That means that people are more productive if they have a very short deadline because they will focus on what they think will delivers value.This creates a change in the working conditions that we will discuss in next part. If the deadline is too short that can be bad for the future of the project, because some activities which are critical for the efficient operations of the project, such as risk management, code documentation, comparison of available COTS solutions or research of the most adapted technologies for the project. On the other hand, if there is too much time allocated to theses sides activities it may take too much time for the value they bring. The aim is to find the right equilibrium between the support activities and the value delivering activities.

Although, some of these parameters can be taken into account using the effort multipliers. But not all such as the use of a weekly project meeting or a daily meetings, even though the COCOMO II has a "site communication" parameter. Theses changes can influence the way the programmer's works by postponing the moment the team will communicate, which may influence the rework programmer do.

The more frequent are releases, the more frequent are the deadline for the organization, so the developers are more efficient. Even though, there might be internal milestones, a release is a stronger deadline because it's not an internal deadline but an external one and missing it is less likely because there is more pressure!

2) Different working styles

The different software lifecycle are not compatible with any software release frequency. Also, changing the release frequency may change the way developers work.

As we see in [16], the release frequency makes developers change the way they communicate and test and integrate the products.

a) Communication

Most of the projects with a lower than quarterly release frequency have weekly or biweekly project meetings. As the frequency of the releases increase, the frequency of the meetings increases too to allow the stakeholders to communicate. The project with monthly or weekly release usually have daily standing meetings. But if the software is released daily, the meetings should be eliminated entirely and the communication should be made online or in real-time.

b) Testing

Testing is a really crucial phase of the development because the release of a version with defects would be a real problem. The testing process needs to be done to ensure the quality of the release.

The project with a yearly release lifecycle or more do usually the testing process manually. But if the release are made quarterly, automated acceptance testing should help the developers to keep up with the pace. If the release frequency is weekly, the organization should go even farther by making development managers responsible for coding and testing and eliminating a separate QA teams.For coping up with a daily releases lifecycle, the developers may have to use assertions, or other self-validating programming techniques.

c) Integration

If the release lifecycle is yearly or more, the developers can do build/integration as a discrete project phase. But if the release frequency is quarterly, the developers should do a continuous integration. Alsoif the release frequency is weekly, the organization should eliminate separate patch release.

C. Business oportunity:

Each release is an opportunity to sell the software to the customer, because some customers like to buy the software when they can have new features, although some others want to stay with an old version of the software for reliability or because they don't want to change the habits of theirs collaborators. But releasing gives visibility to the product so is a business asset.

D. Maintainability:

The release frequency has a strong impact on the number of versions to maintain, for organizations which maintain their software for two year, having a quarterly or yearly cycle change the number of versions to maintain from two to eight. If when adopting a high frequency release, no step is taken not to have too many versions of the software to maintain, the maintenance will become very complex and so costly.

There are several techniques [10] to avoid having too high maintenance costs. The two main options are either to force the user to go to the last version or to have a dual release trains.

We can force the customer to go to the last versions by several means, for instance in case of SaaS software the editors controls the software in a centralized place. Also the user can use automatic updates without notifying the user, like with the Google chrome web browser [11]. Also the software can ask the user if he wants to update, such as Evernote does. If the user encounters any issues with an old release he is advised to go to the last version of the software. It's a very efficient way to deal with updates in the case a personal software, but it may not be optimal for the business applications as we discussed in the influence on the customer part.

The second option is to use dual release trains. That means that there are two kinds of releases, the ones with a long maintenance period and the ones with a shorter ones. We can give the example of the popular Linux operating system Ubuntu which does a bi yearly release [12]. The short maintenance versions have a year of maintainability, which means that the user has six month to update his system after the next version is released. The long term support version have a 4 year maintenance and are release every two years. This allows the user who wants reliability from the system to have a version which has been tested by users for two years and the early adopters to get the last features every 6 months. Also the developers don't have to maintain too much versions.

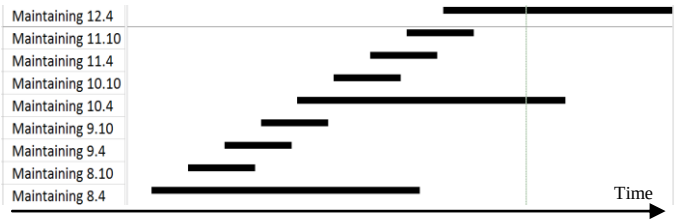

Figure 15 - Gantt chart of Ubuntu maintenance version

As we see in figure 15, the Ubuntu developers have to maintain at most 2 long term versions and 2 normal versions, which is good given the important release and the long term support of some of the versions.

V. CONCLUSION

We saw the influence of the release frequency quantitavely and qualitatively. We modeled the release of software which is a usually modeled as a continuous function by a dicrete serie. First of all, we studied how to create the size of the best quality given schedule and budget. After that we focus on the economic part of the problem to model how to optimize the profit. Then we analized this function by seing the influence of the differents parameters and studying the likelihood of the behavior of the model. In the last part we studied the qualitative effects for the programmers, business and

maintainability. The main changes for the programmers are the influence of Parkinson's law and the effect on way people deal with communication, testing and integration. We looked at the influence on customers, business and maintainability.

REFERENCES

[1] http://blog.chromium.org/2010/07/release-early-release-often.htmlJ

[2] http://mozilla.github.io/process-releases/draft/development_overview/

[3] http://blogs.technet.com/b/microsoft_blog/archive/2013/03/26/looking-back-and-springing-ahead.aspx

[4] http://arstechnica.com/information-technology/2013/01/ubuntu-considers-huge-change-that-would-end-traditional-release-cycle/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+arstechnica%2Findex+%28Ars+Technica+-+All+content%29&utm_content=Netvibes

[5] Vidyanand Choudhary, "Software as a Service: Implications for Investment in Software Development"

[6] http://bits.blogs.nytimes.com/2013/01/29/for-rent-the-new-microsoft-office/

[7] "COCOMO II Model Definition Manual"

[8] Abhijit Dubey and Dilip Wagle ,"Delivering software as a service" in The McKinksey Quarterly May 2007

[9] Mehwish Nasir in "A Survey of Software Estimation Techniques and Project Planning Practices"

[10] http://damonpoole.blogspot.kr/2008/01/tuning-frequency-of-your-releases.html

[11] https://support.google.com/chrome/answer/95414?hl=en

[12] https://wiki.ubuntu.com/Releases

[13] http://redmondmag.com/articles/2007/10/01/word-2007-not-exactly-a-musthave.aspx

[14] http://news.softpedia.com/news/Less-Clicks-to-Launch-Apps-on-Windows-8-than-on-Windows-7-Microsoft-314048.shtml

[15] http://thetechblock.com/evolution-merging-ios-osx/

[16] http://www.thpii.com/2012/02/release-frequency-and-the-adoption-of-agile-practices/