Optimal Interval Time for Enterprise (Business Intelligence) Software Upgrade

I. N. Bisono[1], H. Soewandi[2]
[1]Department of Industrial Engineering, Petra Christian University, Surabaya, Indonesia
[2]Microstrategy, Inc., 1850 Towers Crescent Plaza, Tysons Corner, VA 22182, USA
(mlindri@petra.ac.id, hsoewandi@microstrategy.com)

Abstract - **In this paper, we discuss a situation of enterprise software upgrade that is common in real life. We started with a simplistic model with one software vendor and then multiple software vendors. This model led to an optimal interval time for upgrades that resembles the optimal time in Economic Order Quantity. A more realistic model with discrete time was proposed by adopting MicroStrategy case in releasing their newer software, namely one major upgrade followed by 3 minor upgrades in a year. We proved that the discrete cost function is convex. From an analysis of several numerical examples, we found very interesting and a bit counter intuitive observation.**

Keywords - **Software upgrade, optimal interval time**

I. INTRODUCTION

Enterprise software refers to computer programs used by organizations rather than individual users. It typically includes various business applications, tools for modeling the organization's operations, and development tools for creating custom applications. Business Intelligence (BI) is a type of enterprise software that helps improve decision-making through fact-based support systems.

This paper focuses on determining the optimal interval for upgrading an enterprise software suite, specifically in the context of Business Intelligence software like MicroStrategy Platform Analytics. In typical enterprise scenarios, customers maintain two parallel systems during the upgrade process: a stable production system using an older software version and a user acceptance testing (UAT) system using a newer version. The upgrade involves comparing these systems.

Enterprise software like MicroStrategy often provides testing tools, such as MicroStrategy Integrity Manager, to aid the upgrade process. However, it is also important to involve business users in validating the user experience to ensure that upgrading to the newer version is beneficial.

The main challenge addressed in this paper is finding the right balance for how frequently a customer should upgrade their system. The authors develop a simple mathematical model using both continuous and discrete time assumptions to determine the optimal upgrade frequency for enterprise BI software. Enterprise software refers to computer programs used by organizations rather than individual users. It typically includes various business applications, tools for modeling the organization's operations, and development tools for creating custom applications. Business Intelligence (BI) is an enterprise software that helps improve decision-making through fact-based support systems.

This paper focuses on determining the optimal interval for upgrading an enterprise software suite, specifically in the context of Business Intelligence software like MicroStrategy Platform Analytics. In typical enterprise scenarios, customers maintain two parallel systems during the upgrade process: a stable production system using an older software version and a user acceptance testing (UAT) system using a newer version. Therefore, the upgrade involves comparing these systems.

Enterprise software like MicroStrategy often provides testing tools such as MicroStrategy Integrity Manager to aid the upgrade process. However, it is also important to involve business users in validating the user experience to ensure that upgrading to the newer version is beneficial.

The main challenge addressed in this paper is finding the right balance for how frequently a customer should upgrade their system. The authors develop a simple mathematical model using continuous and discrete time assumptions to determine the optimal upgrade frequency for enterprise BI software.

II. LITERATURE REVIEW

Surprisingly, there are minimal literature reviews that we can find on the subject of software upgrades, not to mention: enterprise software. One of the most relevant pieces of literature we can find is by Vaniea and Rashidi [1]. They surveyed 307 respondents and concluded that users balance the risks and costs of updating against potential benefits. They also suggest several factors to consider, such as (1) information about the newer version, (2) resources that need to be allocated, and (3) the recovery path. However, they do not suggest any particular upgrade interval to follow.

Labuschagne et al. [2] investigated the cost and benefit of automated regression testing on 61 Java projects using the Continuous Integration service provided by GIT-Hub. They found that it was only sometimes clear to have the benefit of a regression test compared to the cost of writing, maintaining, and executing it. Their subject is different from our focus.

Bala and Carr [3] perhaps is the closest to our research. They presented mathematical models that relate the software upgrade to the price. Their research objective is to understand whether offering a special upgrade price is an effective strategy. They concluded that offering a discount is beneficial when the new version has many new functionalities or minor upgrades (since the users will need to be given the incentives to adopt the latest technology). However, when the new version has some new functionalities, there may be better strategies than discounted prices. Nevertheless, they focus on merit, not timing, and they focus on personal software.

Khoo and Robey [4] are one of the research papers (that we can find) that study enterprise software upgrades (SAP and Windows) – they called these "package software." They conducted qualitative empirical research in a large enterprise. The motivation for the upgrade is mainly because this packaged software is near the end of its life cycle (*i.e.*, no longer supported by vendors). They concluded that a software upgrade is a unique type of IS project, with characteristics distinguishing it from maintenance, traditional system development, and initial adoption of a commercial system. In addition, their study found that sunset dates can be a dominating influence on an upgrade decision. Our focus differs from theirs since we try to balance various costs well.

Planning, S. [5] wrote a report for NIST on the economic impacts of inadequate infrastructure for software testing. He presented various economic models for software testing, focusing on insufficient testing. Again, a fascinating topic, but utterly different from ours.

With these findings, we decided to build our mathematical model to understand better. We develop the model based on the seminal work on inventory control that balance between set-up (testing) cost and holding (upgrade testing) cost (see Erlenkotter [6] for detail).

III. METHOD AND DISCUSSION

A. *Costs of Enterprise (BI) Software Upgrade and Simple Mathematical Model*

In this section, we will discuss the cost model that we consider for our optimization. We start with the assumption that a director or executive has a planning horizon *T* unit of time (*e.g.*, ten years, 12 years, 15 years, 20 years, *etc.*) in their mind. For the sake of our discussion, which is a common practice in the industry, when the user wants to upgrade the system, they will need to set up a parallel User Acceptance Testing (UAT) system that needs to be compared with the existing production system.

To do testing, we need to prepare a UAT system: several Computer/Machine/Virtual Machines with enough RAM and hard disk, warehouse database (WH RDBMS) for testing data, etc. Let's assume the set-up cost to prepare all of these hardware and test data is US\$ *K*. Therefore, if we upgrade the Enterprise (BI) system every *t* unit of time (where *t* is our decision variable), we will encounter the following set-up cost for the entire planning horizon:

$$\frac{K \times T}{t}$$

It is typical that once the UAT system is up and running, the BI team (or IT department) will perform automated testing for some benchmark reports. MicroStrategy, as an enterprise (BI) software provider, has a tool such as MicroStrategy Integrity Manager that can be used to compare SQLs and Data for various reports between different versions. This is usually the first step in the software upgrade testing. We can safely assume that the cost of performing this automated comparison is a fixed cost of US\$ $\boldsymbol{h_0}$ as well. Therefore, if the entire planning horizon is *T* unit of time, and we perform upgrade every *t* unit of time, the cost will be

$$\frac{h_0 \times T}{t}$$

Once we have passed the first stage of upgrade testing, we will invite business users to verify and confirm that the UAT system produces the same result as the current production system, and to see whether business users like the look and feel of the version. This type of software system is generally quite challenging and can be costly because there are more business/end users' involvement.

Several literatures in software testing suggested that *n*-way combinatorial testing is one of the most effective way in testing [7, 8]. Some researchers even suggest various techniques on how to generate test sets [9, 10, 11].

Now, imagine that the BI software vendor introduces the same number of new features every time period (say: 1 new feature), and the software testing upgrade will do 2-way testing (test every 2 combinations), then with the assumption that the current system has already used 2 features, the software testing upgrade will need to consider combination sequence $\{{}^3C_2,\ {}^4C_2,\ {}^5C_2,\ {}^6C_2, \ldots\} = \{3, 6, 10, 15, \ldots\}$ of test cases, that means:

- If we upgrade in the next 1 time unit, we need to test 2 combinations out of (2+1=3) features.
- If we upgrade in the next 2 time unit, we need to test 2 combinations out of (2+2=4) features.
- If we upgrade in the next 3 time unit, we need to test 2 combinations out of (2+3=5) features.
- *Etc.*

Therefore, *n*-way combinatorial software testing sequence above suggests that the software upgrade testing will be a function of our decision variable *t* (optimal inter-upgrade time). If we wait for 1 time-unit to upgrade for the new version, we will need to run 3 sets of test cases. If we wait for 2 time-units, we will need to perform 6 sets of test cases. Similarly, if we wait for 3 time-units, we will need to do 10 test cases, *etc*. Notice that this sequence can be easily proven to be convex (see equation 4 below). Therefore, without loss of generality, we can define a slightly conservative approach to model the software upgrade testing cost as a linear function (or any convex function to approximate this cost):

$$h_1 + h_2 \times t$$

Hence, in our model, the total cost (*TC*) for the entire planning horizon (*T*) will be given by:

$$TC = \frac{(K + h_0) \times T}{t} + h_1 + h_2 \times t \ (T \geq t \geq 0, \forall t \in R) \qquad (1)$$

Notice that *TC* is a convex function since its summons are convex respectively. Therefore, taking the derivative of *TC* with respect to *t*, we get the optimal inter-upgrade time to be:

$$t^* = \sqrt{\frac{(K+h_0)\times T}{h_2}} = \sqrt{\frac{K_1\times T}{h_2}} \qquad (2)$$

It is very important for us to understand the interpretation of the software upgrade testing cost h_2 in reality. This is the cost to invite business (end) users to validate the newer version, and it really depends upon how often we perform an upgrade (or how different the current production version to the newer version). In practice, this is usually an estimate based on the amount of business (end) users that participate in the pilot project for the UAT system. It also does not mean that those business (end) users will do user acceptance testing for the entire duration of *t* time unit. We suggest using the compensation rate of those business (end) users per unit time multiply by a percentage.

Many who are familiar with inventory control would quickly point out that the optimal inter-upgrade time in (2) resembles the Economic Order Quantity (EOQ) of Harris [12]. While the optimal solution in (2) looks very simple, it can explain many real-life phenomena in the industry, and it can also be expanded to cover more complicated cases, such as more than one software needs to be upgraded, *etc.*

Let's now consider a few critical aspects of those simple formulas. Consider an organization with a fixed planning horizon $= T$ (say: 10 years), and analyze the K_1/h_2 ratios. It is straightforward to see that when the set-up (fixed) cost per testing is relatively high to the cost of bringing business (end) users to help validate the newer version, the organization tends to be conservative in their upgrade. Conversely, suppose an organization tends to bring more of their business users into the user acceptance testing, *i.e.* the K_1/h_2 ratio becomes small (since h_2 is higher), then the inter-upgrade time should be shorter, *i.e.*, the organization should upgrade more often. It is very interesting to point out that this simple phenomenon does not seem true in real life. Many BI directors/executives for large enterprises are afraid to do frequent updates.

When the enterprise (BI) software provider observes the above phenomenon, naturally, the reaction is to modify the cadence of the release software. Many enterprises (BI) software providers change their release cadence into a much more predictable cadence with minor releases to help and encourage customers to upgrade to the new versions and, more importantly, release software as Service Pack (or Update or Minor release) in an effort to reduce the *n*-way combinatorial testing cost. This leads to a more realistic model below.

B. *More Realistic Model for Optimal Upgrade Interval*

As we have discussed in the previous section, many enterprise (BI) software vendors, *e.g.*, MicroStrategy, actually adopted a more predictable cadence in releasing their newer software with major release follows by minor (service packs or update releases) such as:

- Dec 2018: MSTR 2019 GA (major release)
- Mar 2019: MSTR 2019 Update 1 (minor release)
- Jun 2019: MSTR 2019 Update 2 (minor release)
- Sep 2019: MSTR 2019 Update 3 (minor release)
- Dec 2019: MSTR 2020 GA (major release)
- Mar 2020: MSTR 2020 Update 1 (minor release)
- Jun 2020: MSTR 2020 Update 2 (minor release)
- Sep 2020: MSTR 2020 Update 3 (minor release)
- Dec 2020: MSTR 2021 GA (major release)
- Mar 2021: MSTR 2021 Update 1 (minor release)

In term of software upgrade testing cost, the above release cadence can be depicted to have the following cost consequence as in Figure 1 (remember that the software upgrade testing cost essentially affected by how many new features the software version has, as we have pointed as the result of *n*-way combinatorial testing).

In this situation, the set-up cost is more or less the same. However, we will need to discretize the decision variable, *i.e.*, inter-upgrade time (*t*) and adjust the upgrade testing cost accordingly. For simplicity, we can break upgrade testing cost into two different upgrade testing costs, namely: upgrade testing cost for major release (h_1/quarter) and upgrade testing cost for minor release (h_2/quarter). It makes sense, and without loss of generality, if we assume: $h_1 \geq h_2$ since testing for major release will have more new codes. For the decision variable (*t* time unit), it can now be defined as: how many quarters (a discrete number of quarters) from the last update.

Now, the Total Cost function in (1), for $\forall t \in Z$ and $t \geq 1$, can be rewritten as:

$$\begin{aligned} TC(t) &= \frac{K_1\times T}{t} + \sum_{n=1}^{t} h(n) \\ &= \frac{K_1\times T}{t} + h_2 \times t \; + (h_1 - h_2) \times \left\lfloor \frac{t-1}{4} \right\rfloor + (h_1 - h_2) \end{aligned} \qquad (3)$$

Please note that we write the total cost in (3) that way for the purpose to establish the proof. It should be clear that $\forall t \geq 1$, we have: $\sum_{n=1}^{t} h(n) = h_2 \times t + (h_1 - h_2) \times \left\lceil \frac{t}{4} \right\rceil = h_2 \times t + (h_1 - h_2) \times \left\lfloor \frac{t-1}{4} \right\rfloor + (h_1 - h_2)$.

Following Murota (2008, 2015), we define a univariate function f: $Z \to R$ as a discrete convex function if:

$$f(t-1) + f(t+1) \geq 2f(t) \quad (\forall t \in Z) \qquad (4)$$

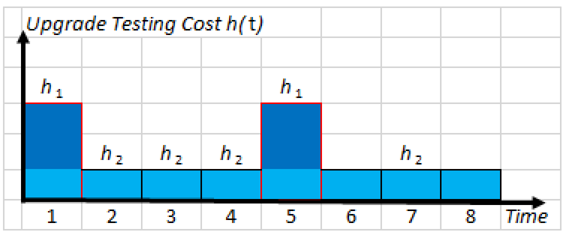


Fig. 1. The upgrade testing cost will be the area under the above curve

Lemma 1:
If $mod(t, 4) = 2, 3,$ and 0, then the total cost *TC*(*t*) in (3) is a discrete convex function.
Proof:

The proof is straight forward, we simply use the definition in (4) and the fact that

$\left\lfloor \frac{x}{y} \right\rfloor = \frac{x-mod(x,y)}{y}$ where *mod*(*x*,*y*) is the remainder of $\frac{x}{y}$

Then enumerate for *t* = 2, 3, 4, Here is the detail. First the right hand side of the inequality in (4):

TC(*t* – 1) + *TC*(*t* + 1) =

$$\begin{pmatrix} \frac{K_1\times T}{(t-1)} + h_2 \times (t-1) + \\ (h_1-h_2)\times\left\lfloor\frac{t-2}{4}\right\rfloor + (h_1-h_2) \end{pmatrix} + \begin{pmatrix} \frac{K_1\times T}{(t+1)} + h_2 \times (t+1) + \\ (h_1-h_2)\times\left\lfloor\frac{t}{4}\right\rfloor + (h_1-h_2) \end{pmatrix}$$

$$= \frac{2\times K_1\times T\times t}{(t^2-1)} + 2\times h_2\times t + (h_1-h_2)\times\left(\left\lfloor\frac{t-2}{4}\right\rfloor + \left\lfloor\frac{t}{4}\right\rfloor\right) + 2\times(h_1-h_2)$$

$$= \frac{2\times K_1\times T\times t}{(t^2-1)} + 2\times h_2\times t + (h_1-h_2)\times \left(\frac{(t-2)-mod(t-2,4)}{4} + \frac{t-mod(t,4)}{4}\right) + 2\times(h_1-h_2)$$

$$= \frac{2\times K_1\times T\times t}{(t^2-1)} + 2\times h_2\times t + (h_1-h_2)\times \left(\frac{2t-mod(t-2,4)-mod(t+1,4)}{4} - \frac{1}{2}\right) + 2\times(h_1-h_2) \qquad (5)$$

Now, the left hand side of the inequality (4) can be re-written as:

$$2 * TC(t) = 2\times\left(\frac{K_1\times T}{t} + h_2\times t + (h_1-h_2)\times\left\lfloor\frac{t-1}{4}\right\rfloor + (h_1-h_2)\right)$$

$$= \frac{2\times K_1\times T}{t} + 2\times h_2\times t + 2\times(h_1-h_2)\times\left(\frac{t-mod(t-1,4)}{4} - \frac{1}{4}\right) + 2\times(h_1-h_2) \qquad (6)$$

Since *mod* is a cyclical function, we just need to compare equations (5) and (6) for *t* = 2, 3, and 4. That is we need to prove that

$$\frac{2\times K_1\times T\times t}{(t^2-1)} - (h_1-h_2)\times\left(\frac{mod(t-2,4)+mod(t+1,4)}{4}\right) \geq \frac{2\times K_1\times T}{t} - 2\times(h_1-h_2)\times\frac{mod(t-1,4)}{4}$$

- For *t* = 2, we have $\frac{4\times K_1\times T}{3} \geq K_1\times T$
- For *t* = 3, we have $\frac{3\times K_1\times T}{4} \geq \frac{2\times K_1\times T}{3}$
- For *t* = 4, we have $\frac{8\times K_1\times T}{15} \geq \frac{K_1\times T}{2}$

Q.E.D.

Unfortunately, we cannot easily establish convexity of the cost function tor *t* = 5, 9, 13, 17, … However, we can have the following: Define transformation $t = 4\times s + 1$ ∀*s*∈ *Z* and *s* ≥ 1, and we have the following Lemma.

Lemma 2:

If $t = 4\times s + 1$, ∀*s*∈ *Z*, *s* ≥ 1, then the total cost *TC*(*s*) in (3) is a discrete convex function with respect to *s*.

Proof:

With the transformation, we have the following cost function: $TC(s) = \frac{K_1\times T}{4s+1} + h_1 s + 3h_2 s$. Applying the definition of discrete convex function in (4), we have:

$$\text{LHS} = \frac{K_1\times T}{4(s+1)+1} + h_1(s+1) + 3h_2(s+1) + \frac{K_1\times T}{4(s-1)+1} + h_1(s-1) + 3h_2(s-1) = \frac{K_1\times T}{4s+5} + \frac{K_1\times T}{4s-3} + 2h_1 s + 6h_2 s$$

$$= \frac{(8s+2)(K_1\times T)}{(4s+5)(4s-3)} + 2h_1 s + 6h_2 s$$

and

$$\text{RHS} = \frac{K_1\times T}{8s+1} + 2h_1 s + 6h_2 s$$

Since $K_1\times T > 0$, it is straight forward to see that

$\frac{(8s+2)(K_1\times T)}{(4s+5)(4s-3)} \geq \frac{(K_1\times T)}{8s+1}$, ∀*s*∈ *Z*, *s* ≥ 1 q.e.d.

Algorithm to Find the Optimal Upgrade Interval

With the above Lemma 2 and Lemma 1, we can then design the following simple algorithm to find the optimal inter-upgrade time.

Algorithm:

```
Initialize TC(t = 1) = according to $K_1 \times T + h_1$
Apply Lemma 2 for t = 5, 9, …
        Calculate TC(t) according to (3)
        If TC(t) > TC(t – 4), break
Apply Lemma 1 to search the minimal TC for (t – 1), (t – 2), (t – 3) OR (t + 1), (t + 2), (t + 3)
```

It is worth it to point out that the total cost *TC*(*s*) = $\frac{K_1\times T}{4s+1} + h_1 s + 3h_2 s$ is actually the total cost when we consider yearly cost (instead of quarterly cost). Since the behavior of enterprise (BI) software vendor is to produce major release at the beginning of the year, the total cost for the entire year is constant. Hence, the problem can actually be reduced to the original problem in (1) which is convex.

C. *Numerical Examples and Observations*

To have a better understanding of the difference on enterprise (BI) software upgrade between countries where labor cost of upgrade testing software is relatively cheaper than the set-up cost *vs.* countries where labor cost is expensive relative to the set-up cost, we simulated the total cost (Eq. 3) with various combination ratio h_1 on h_2 and K_1 on h_2.

For the simulation, we use planning horizon (*T*) of 10 years (= 40 quarters), and h_2=1. Obviously, in the situation when the set-up cost (K_1) is relatively high compare to the upgrade testing cost (h_2), then the adoption of newer version of the enterprise (BI) software will be longer. Take for example, when K_1 ten times h_2 and h_1 = h_2, the simulation resulted optimal inter-upgrade times of 20 quarters. On the other hand, when the set-up cost is less dominant, it is cheaper to upgrade sooner.

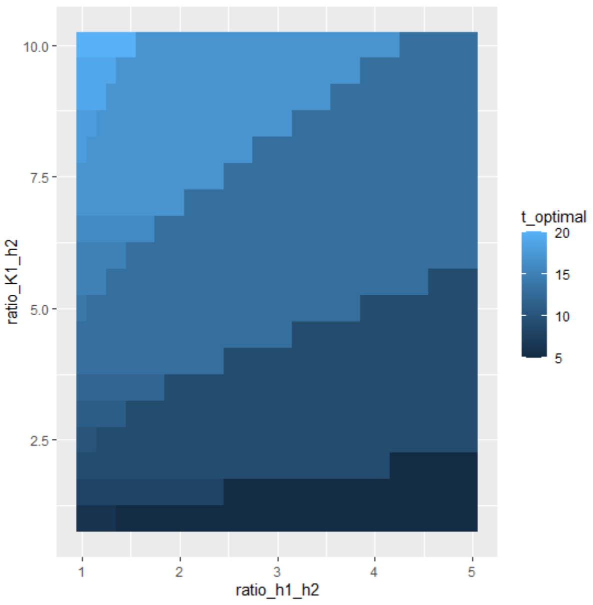
The simulation result grid shown that when the set up cost is twice at much the minor set-up cost, the optimal time for upgrade. This observation is a bit counter intuitive from the action of the enterprise software vendor. The explanation for this is that the vendor is very likely a more dominant where its software is very much needed and the new functionalities that are offered very much needed as well.

IV. CONCLUSION

In this paper, we presented a simple mathematical model that is very similar to the famous economic order quantity from Harris. We use the argument from *n*-way combinatorial testing to model the software testing upgrade when an enterprise upgrades its enterprise (BI) software. Even though, simple model, our model can be used to explain some real-life scenarios.

We further enhance our mathematical model to account for more realistic scenario by accounting for the release schedule of the enterprise (BI) software, and discretize the time period. We further develop a very simple numerical computation algorithm for planning purpose. Furthermore, our numerical simulation found very interesting and a bit counter intuitive observations. This by itself may need to be further researched.

Clearly, our optimal inter-upgrade system mathematical model for enterprise software can also be further enhanced into several different directions, *e.g.*, when the release version for every software follow different schedules (RDBMS software vendor has its own release schedule, Operating Systems, Mobile, and various other software may also have different release schedule and their own variations in term of functionalities).



REFERENCES

[1] Vaniea, Kami and Yasmeen Rashidi (2016), “Tales of Software Updates: The process of updating software”, CHI 2016: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, May 2016, pp. 3215 – 3226.

[2] Adriaan Labuschagne, Laura Inozemtseva, and Reid Holmes (2017), “Measuring the Cost of Regression Testing in Practice: A Study of Java Projects using Continuous Integration”, ESEC/FSE 2017: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, August 2017, pp. 821–830.

[3] Ram Bala and Scott Carr (2009), “Pricing Software Upgrades: The Role of Product Improvement and User Costs”, Production and Operations Management, Vol. 18 No. 5, September – October 2009, pp. 560–580.

[4] Huoy Min Khoo and Daniel Robey (2007), "Deciding to upgrade packaged software: a comparative case study of motives, contingencies and dependencies", European Journal of Information Systems, Vol. 16, pp. 555–567.

[5] Planning, S. (2002). The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*.

[6] Donald Erlenkotter (2014), “Ford Whitman Harris's Economical Lot Size Model”, International Journal of Production Economics, Volume 155, September 2014, pp. 12 – 15.

[7] Sangeeta Sabharwal and Manuj Aggarwal (2017), "A novel approach for deriving interactions for combinatorial testing", Engineering Science and Technology, Vol 20, pp. 59 – 71.

[8] Bryce, R. C., Lei, Y., Kuhn, D. R., & Kacker, R. (2010). Combinatorial testing. In *Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization* (pp. 196-208). IGI Global.

[9] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence (2007), "IPOG: A General Strategy for T-Way Software Testing," 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), 2007, pp. 549-556.

[10] Yu, L., Duan, F., Lei, Y., Kacker, R. N., & Kuhn, D. R. (2014, January). Combinatorial test generation for software product lines using minimum invalid tuples. In *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering* (pp. 65-72). IEEE.

[11] Abdullah B. Nasser, Yazan A. Sariera, AbdulRahman A. Alsewari, and Kamal Z. Zamli (2015), "Assessing Optimization Based Strategies for t-way Test Suite Generation: The Case for Flower-based Strategy", 2015 IEEE International Conference on Control System, Computing and Engineering, 27 - 29 November 2015, Penang, Malaysia.

[12] Harris, F.W. (1915), "What quantity to make at once", The Library of Factory Management, Vol. 5, Operation and Costs, A.W.ShawCompany, Chicago, pp.47 – 52.

[13] Murota, Kazuo (2009), “Recent Developments in Discrete Convex Analysis, Research Trends in Combinatorial Optimization”, Bonn 2008 (W. Cook, L. Lovasz and J. Vygen, eds.), Springer- Verlag, Berlin, Chapter 11, pp. 219 – 260.

[14] Murota, Kazuo (2015), “Discrete Convex Analysis”, Hausdorff Institute of Mathematics, Summer School (September 21 – 25, 2015).