Software Engineering
2013; 1(2): 7-12
Published online September 10, 2013 (http://www.sciencepublishinggroup.com/j/se)
doi: 10.11648/j.se.20130102.11

Cost models and productivity building applications based on the notification of changes in databases

Jose-Ramon Coz-Fernandez, Ruben Heradio-Gil, Jose-Antonio Cerrada-Somolinos

Departamento de Ingeniería de Software y Sistemas Informáticos. Universidad Nacional de Educación a Distancia. Ciudad Universitaria, Juan del Rosal 16, E-28040. Madrid, Spain

Email address:
jrcozf@gmail.com(Jose-Ramon Coz-Fernandez), rheradio@issi.uned.es(Ruben Heradio-Gil), jcerrada@issi.uned.es(Jose-Antonio Cerrada-Somolinos)

Tocite thisarticle:
Jose-Ramon Coz-Fernandez,Ruben Heradio-Gil, Jose-Antonio Cerrada-Somolinos. Cost Models and Productivity Building Applications Based on the Notification of Changes in Databases.*Software Engineering.*Vol. 1, No. 2, 2013, pp. 7-12. doi: 10.11648/j.se.20130102.11

Abstract: This paper presents a generative approach to build a Software Product Line (SPL). This Software Product Line is used to build applications based on the Notification of Changes in databases. The paper highlights the benefits, in terms of productivity and cost, using this approach. To obtain the economicdata we have used two cost models, the SIMPLE Model (Structured Intuitive Model for Product Line Economics) and one adaptation of COPLIMO (Constructive Product Line Investment Model). Both models demonstrate the great productivity of this SPL. The paper also introduces the Exemplar Driven Development (EDD) process used to build the Software Product Line.

Keywords: Databases, Domain Engineering, Generative Programming, Software Product Lines, Cost Models

1. Introduction

Some of the more significant research lines advocating for the increase in the productivity of the software development are the Generative Programming (GP) [1] and Software Product Lines (SPL) engineering [2]. GP proposes to raise the level of abstraction of programming languages through specifications or models. Early case studies have exhibited significant barriers to adopt an SPL [3] approach. The approach proposed in this paper is the construction of a SPL using an approach based on GP. To build the SPL an adaptation of the [4] Exemplar Driven Development (EDD) is used. The SPL created is about the notification of changes in databases [5].

The purpose of the notification of changes in databases is to provide a range of services to users to make them aware of the changes that are being produced in a database.The paper presents two Cost Modelsfor this SPL based in the SIMPLE [6] [7] [8] and the COPLIMO [9] (Constructive Product Line Investment Model). These models show some analysis about the high productivity and profitability of the SPL proposed. Researchers at the Software Engineering Institute (SEI), Clemson University, the Fraunhofer Institute for Experimental Software Engineering, and Siemens created the Structured intuitive Model for Product Line Economics (SIMPLE), and the COPLIMO is a COCOMO [10] extension.

This paper is structured as follows: Section 2 presents the domain and describes the generative approach. Section 3 summarizes the SIMPLE Model, presents the Cost Model for this approach and offers different sceneries for the Cost Model application. Section 4 gives an overview of the COPLIMO approach for this SPL. The solution provided in several study cases and the comparative with both models are presented in Section 5. Finally, Section 6 summarizes the presented work.

2. Software Product LineOverview

The research problem is to find the Cost Model for implementing the changes notification service (CNS) in databases, using a generative approach. The CNS isresponsible for the communicationof the changes that happen in the database to the subscribed users or systems. Users or systems can be interested only in specific events. For instance, they may need to be reported about: insertions, deletions, updates, logins, logouts, startups, shutdowns and others.To implement this kind of features nowadays databases offer different mechanisms such as Advanced Queue, Pipes and Alert / Signals technologies, different procedural extension languages such as PL / SQL and specialized libraries that extend these languages such as AQ, Pipe and Alert /Signals Libraries [11].

Although these utilities facilitate the developments, products must be programmed manually and the cost of development is high. The development of specific products for this domain depends not only on the specific requirements established (priorities, time management, subscribers, searches, granularity of the solution, visibility, navigation between messages and so on), but the internal structure of the database (tables, keys, users and others). The requirements of this domain have been analyzed after developing several products.

A new DSL (Domain Specific Language), called Notification Change Service Language (NCSL) has been developed to gather the domain variability and specify the domain requirements. This language is expressed in BNF (Backus-Naur form) notation. All the requirements of our domain are specified through this language. In order to derive new products, the application engineer writes NCSL specifications, from the user requirements, that are completed with information automatically gathered from the database as tables, users, fields, keys, schemas and others. Some elements of this NSCL describe variability related to the internal database information (tables, keys, schemas and others) whereas other elements describe the events priorities, times, subscribers, type of visibility, events to be notified by the service, permissions and so on. Users, through a program implemented for this purpose, specify their needs against this NSCL.

For our generative approach, EDD is used. EDD is a SPL methodology which takes advantage of the similarities among domain products to build them by analogy [10]. The EDD starting point is whatever domain product built using conventional software engineering. The product that must exist as the start point of EDD is called exemplar. It is assumed that this exemplar implements implicitly the intersection of all the domain product requirements. To satisfy the domain variable requirements that are out of the intersection, EDD uses the concept of exemplar flexibilization. Figure 1 illustrates a summary of EDD.

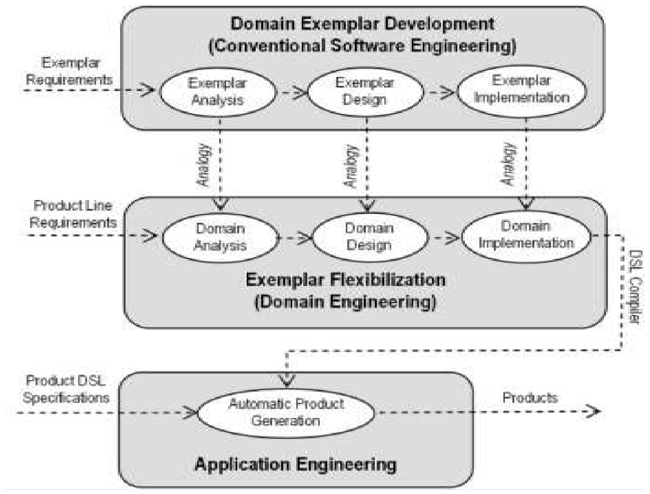


Figure 1.*Summary of EDD.*

The flexibilization is the mechanism that allows establishing an analogy relation (in a formal way) between the exemplar and the new product, so the new products can be derived automatically from the exemplar. The tool that performs the flexibilization is a domain specific compiler (DSC), which is used during application engineering phase to derive automatically new products.An adaptation of EDD has been developed, where a NSCL is built specifying the user features and using the necessary information from the database. This database information is contained in meta-tables and it is obtained automatically. Once the domain specific language exist (in this case, the NSCL), the DSC for this language is implemented.EFL is an external flexibilization technique that supports noninvasive exemplar transformations and crosscutting flexibilizations. It is applicable to whatever kind of software artifact and provides an efficient generative variant construction [12] [13].

EFL is used to build the DSC that deal with the specification variability and also with the implementation variability in our domain. A typical DSC written in EFL is made of an analyzer which takes as input a specification, and a generator which is responsible for generating the new product. The most important part is that generator is responsible for analyzing the exemplar and adapts it in order to generate the new product according to the given specification. In our approach the generator is also responsible for detecting dependencies and inconsistencies in the configuration model.Finally, some sub-generators can analyze the internal elements of the database to obtain all the necessary information of the domain. Figure 2 illustrates how the process works: the analyzer obtains the information from the NCSL (DSL specification) and one generator formed by several sub-generators work coordinated to get the rest of the products of the line.

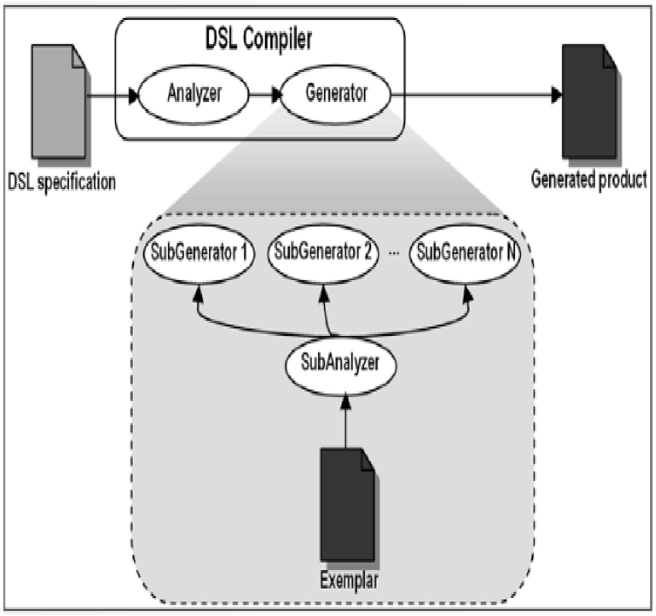


Figure 2.*Generator and Compiler based in EFL.*

3. The SIMPLE approach

Once we have obtained the SPL, our interest is focused to analyze the productivity of our approach. We started using SIMPLE (Structured intuitive Model for Product Line Economics), because it is a model to facilitate decision-making in a product line context by allowing a decision-maker to calculate the costs and benefits of different decision alternatives.

SIMPLE employs a small set of basic cost functions and benefit functions to allow a product line decision-maker to decompose the decision into constituent (and easily valuable) parts.SIMPLE is based on the observation that establishing and then using a product line engineering capability involves the following four costs:

- C_{ORG}: The cost to an organization of adopting the product line approach for its products. Such costs can include reorganization, process improvement, training, and whatever other organizational remedies are necessary.
- C_{CAB}: The development cost to develop a core asset base suited to support the product line being built. This includes costs such as commonality/variability analysis, a generic software architecture, and the cost of developing the software and its supporting designs, documentation, and test infrastructure.
- C_{UNIQUE}: The development cost to develop unique software that is not based on a product line platform. Usually this will be a small portion of a product but in the extreme it could be a complete product.
- C_{REUSE}: The development cost to reuse core assets in a core asset base. This includes the cost of locating and checking out a core asset, tailoring it for use in the intended application (if necessary), and performing the extra integration tests associated with reusing core assets.

The cost of establishing a product line consisting of n products can be written as

$$C_T = C_{ORG}+C_{CAB} + \sum_{i=1}^{n}(C_{UNIQUE}(product_i) + C_{REUSE}(product_i)) \quad (1)$$

Where C_{ORG}, C_{CAB}, C_{UNIQUE} and C_{REUSE} are cost functions that, given the appropriate parameters, return the corresponding costs.For our generative approach, and using the SIMPLE Cost Model, we have the next assumptions:

$$C_{ORG} = 0;\ C_{UNIQUE} = 0;$$

$$C_{CAB} = \sum_{i=1}^{n}(C_{TEST}(product_i)) + C_{DSL} + C_{DSLC} + C_{EX} (2)$$

Where:

- C_{DSL}: The cost to develop the Domain Specific Language (NCSLC – Notification Changes Service Language).
- C_{DSLC}: The cost to development the Domain Specific Language Compiler. This is the cost of the exemplar flexibilizations (C_{FLE}) and the cost of the generators development (C_{GEN}).
- C_{TEST}: The cost to develop the initial test products (*n* is the number of test products). This cost is not necessary to be considered for all thecases, however weconsider it because we have developed several initial test products for our SPL.
- C_{EX}: The development cost for the exemplar. Since our exemplar is similar to every product, C_{EX} is like C_{TEST} for one product.

Then the Total Cost of our SPL (C_{TSPL}) is:

$$C_{TSPL} = \sum_{i=1}^{n+1}(C_{TEST}(product_i)) + C_{NCSL} + C_{FLE} + C_{GEN} \quad (3)$$

C_{NCSL} is the cost to specify all the requirements. In our SPL all the requirements have been integrated into the NSCL program and it is not necessary to do the flexibilizations on the exemplar. This cost,C_{NCSL}, is negligible with respect the rest (see the equation 1 for evaluating the NSCL program cost).Then, our Cost Model is:

$$C_{TSPL} = \sum_{i=1}^{n+1}(C_{TEST}(product_i)) + C_{GEN} \quad (4)$$

The Cost of the development of building product independently, with the SIMPLE Model, should be:

$$C_{TSPL} = \sum_{i=1}^{m}(C_{prod}(product_i)) \quad (5)$$

As the number of test products (n) is always less than the number of total products (m), we obtain the first conclusion: *if the Cost of the Generators (C_{GEN}) is not very high then the cost of our SPL is less than the cost of developing the products independently*. The productivity of our SPL depends on the cost of the generators. For our case, we can consider a similar cost for every test product, then:

$$C_{TSPL} = (n+1)C_{prod}(product_i)+C_{GEN} \quad (6)$$

If it is not necessary to develop the initial test products, then n=0. Then, our second conclusion is: *the cost of our SPL is the cost of the generator plus the cost of the exemplar development (similar to one product)*.

In the case of the development of our products independently, the analysis made shows that the cost of every product has dependencies with the database size. The database size, in a simplified form, depends on the number of entities with requirements of notification changes and the number of users or systems who subscribe to the NCS. In all cases the cost of every product can be expressed as:

$$C_{prod}(product_i) = C_1 + NC_2 + UC_2 \quad (7)$$

Where $C_1$and C_2 are constants, N is the number of entities with Notification Changes Requirements and U is the number of subscriptions to the NCS. The constants values depends on the mechanism used to build the products. Then, the cost of the development of all the products is:

$$C_T = \sum_{i=1}^{m}(C_{TEST}(product_i)) = m(C_1 + NC_2 + UC_2) \quad (8)$$

Where m is the number of products. Then, if we compare the C_T (the cost of the development of the products independently) with the Cost of the SPL, we obtain the next conclusion about the SPL cost.

$$C_{GEN} < m(C_1 + NC_2 + UC_2) \Rightarrow C_{TSPL} < C_T \qquad (9)$$

The number of products that can be obtained with the proposed SPL is measured. This number corresponds to the combinations of features that have sense, that is to say, that do not maintain dependencies or constraints among them. If this value is high, the CGEN is very much less than CT. The third conclusion isthe cost of the SPL is much lesser than the cost of development the products independentlyif the number of the products of the SPL is high.

4. TheCOPLIMO approach

We used another different approach based on the standard COPLIMO, a COCOMOIIextension, in order to obtain the productivity of the SPL developed.According to this standard and having to:

- PLS(N) is the Product Line Savings for a Software Product Line (SPL) with N products.
- PMR(N) is the cost in PM (person / months) for building N products in a Software Product Line (SPL)
- PMNR(N) is the cost in PM for building N products without reusing components (outside of the SPL)

We obtain the main equation (10) of our economic model:

$$\mathrm{PLS(N)} = \mathrm{PMNR(N)} - \mathrm{PMR(N)} \qquad (10)$$

Where PMNR(N) is estimated as:

$$\mathrm{PM} = A\mathrm{Size}^{E} \prod_{i=1}^{n} \mathrm{EM}_i \qquad (11)$$

- A is an organization-depend constant.
- E is the “scaling parameter”. It reflects the disproportionate effort for large projects, due to the growth of interpersonal communications overhead and growth of large-system integration overhead.
- $\mathrm{EM_i}$ are Effort Multipliers (required software reliability, database size, product complexity, required reusability…).

If we have the COPLIMO assumptions, PMR(N) is estimated by:

$$\mathrm{PMR(N)} = \mathrm{PMR(1)} + (\mathrm{N}-1)\mathrm{PMNR(1)}\left(\mathrm{PFRAC} + \mathrm{RFRAC}\,\frac{\mathrm{AA}}{100} + \mathrm{AFRAC}\right) \qquad (12)$$

Where:

- PFRAC, RFRAC and AFRAC are the unique, back-box and white-box reused parts of our products.
- RCWR (Relative Cost of Writing for Reuse) is a multiplier to estimate the effort of making software reusable across the SPL.
- AA (Assessment and Assimilation) is the effort required to assess the candidate reusable components and choose the most appropriate one, plus the effort to assimilate the component code and documentation into the new product.

For our Software Product Line (SPL) we have:

$$\mathrm{PMNR(N)} = NA\ \mathrm{esize}^{E} \prod_{i=1}^{n} \mathrm{eEM}_i \qquad (13)$$

Where esize is the size of the exemplar and eEM are the Efforts Multipliers of the exemplar. The “scaling parameter” is:

$$E = B + 0.01 \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{SF_i}\, a \qquad (14)$$

Where B is the “scaling base exponent for the effort” and SFi are the scale factors: precedentedness, development flexibility, architecture, risk resolution, team cohesion and process maturity. Finally, the cost for building *N* products in our SPL is:

$$\mathrm{PMR(N)} = \mathrm{PMR(1)} + (\mathrm{N}-1)\frac{\mathrm{AA}}{100\,\{\sum_{\mathrm{i=1}}^{\mathrm{G}}\{\mathrm{A\ gsize^E} \prod_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{gEM_i}\}\}} \qquad (15)$$

Where gsize is the size of the generators, G is the number of generators and gEM are the Efforts Multipliers of the generators.Our interest is to obtain the Return On Investment (ROI):

$$\mathrm{ROI(N)} = \frac{\text{cost savings}}{\text{cost investment}} = \frac{\mathrm{PLS(N)}}{|\mathrm{PLS(1)}|} \qquad (16)$$

In the paper [14] is showed a specific application for the Notification Services related with ITIL (Technology Infrastructure Library) [15], using a similar approach. We use this COMPLIMO adaptation to compare with our initial model based in SIMPLEfor several study cases discussed in the next section.

4. The SPL and Several Study Cases

The presented SPL has been employed for conducting several case studies with different databases: a database supporting a university that offers courses, a control system for air navigation that contains electronic controls that inform pilots of changes in different aspects of navigation and others. Oracle database has been used in all these case studies. Different implementation mechanisms provided by Oracle are applied for implementing the products: Pipes, Signals and Alerts, Advanced Queue Management. In all of these studies the SPL generates the 100% of the new products, covering all the features specified.

Using the SIMPLE approach, the study of investment profitability is summarized below for 18 products; we used the metrics in LOC (Lines of Code). We used the cost of the generators necessary to develop the SPL and the total cost of the SPL included the test products and the NCSL Program.

Table 1.*Value of the Parameters based in SIMPLE*

Mechanism	Parameters	Values
Pipes	C_1, C_2, m	111, 17, 18
Advanced Queue	C_1, C_2 , m	271, 46, 18
Signals – Alerts	C_1, C_2, m	111, 17, 18
Pipes	C_{GEN}, C_{TSPL}	2500 LOC, 3000 LOC
Advanced Queue	C_{GEN}, C_{TSPL}	2500 LOC, 3000 LOC
Signals – Alerts	C_{GEN}, C_{TSPL}	2500 LOC, 3000 LOC

Then, using the third conclusion discussed in the Section 3, we obtain that in the case of the Pipes or Signals – Alerts mechanisms for only one subscriber to the NCS, if our database has only three entities to develop the SPL, the SPL has more productivity than the development of the products independently. In the case of Advanced Queue mechanism, with only 16 products to develop, our SPL has more productivity than to develop the products independently, using a small database.

In our study cases we have worked with several databases: the first database has the smallest size, with only 8 entities that contained changes notifications features and with 10 subscribers to the notifications changes service. The second one, with a medium size, contained 50 tables and 10 subscribers. The third one contained 200 tables and 30 subscribers and in the last one there are 400 tables and 50 subscribers. With all these data the productivity of our SPL is very high. The number of products obtained in some of these case studies with this mechanism is about tens of thousands. This estimation is calculated using all the valid requirements combinations. The number of valid feature combinations is illustrated in table 2.

Table 2.*Number of Products*

Type or Requirement	Number of combinations
Pipes	2
Time Management	12
Subscriptions	2
Granularity	3
Priority	2
Aggrupation	2
Visibility	3
Navigation	3
Searches	2
Waits	3
Operations	32
TOTAL	497.664
Combinations Not Valid	124.416
Number of Products	373.248
TOTAL	**497.664**

This shows the productivity of our SPL, using the SIMPLE approach.Using the COPLIMO approach, we get all the parameters of our model. Some of them are listed in the table 3.

Table 3.*Value of the Parameters based in COMPLIMO*

Parameter	Description	Value
∑SFj(gen)	Sum of all Scale Factors for the Generators	6.32
∏EMj (gen)	Product of 17 Effort Multipliers for the Generators	2.33
E (gen)	Scaling exponent for effort (Generators)	0.97
AA	Assessment and Assimilation	4

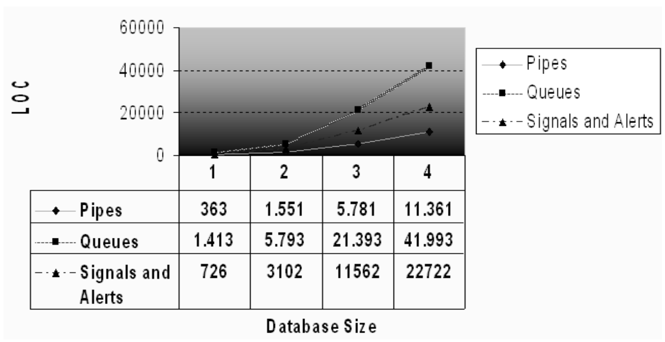
Substituting all parameters into the formulas and using the algorithms described in [16], where N is the number of products in our SPL, we obtain that the number of products necessary for our product line has benefits is more than 10. We used the case of a middle size database (third case of our study cases) and the Advanced Queue mechanism:

$$\mathrm{N} > 10 \Rightarrow ROI(\mathrm{N}) \geq 0 \qquad (17)$$

That is, with only 10 products, our SPL will be productive. In the SIMPLE case we obtained that with only 16 products to develop, our SPL has more productivity than to develop the products independently. This case was the smallest database. In the case of the middle size database we need only 12 products (similar results). In the case of SIMPLE model the data obtained are based in the number of LOC and for the COMPLIMO approach we introduced another data as the "*scaling parameters*", the "*assessment and Assimilation*"parameter or the "*effort multipliers*".

For the larger databases the code to be generated is bigger than for the small ones. The database size, in a simplified form, depends on the number of entities with requirements of notification changes, the number of users of the database who subscribe to the notification changes service and the number of attributes in each entity.

This study shows that the profitability increases with the size of the database, that is to say, more code is automatically generated. In this study we have considered different implementation technologies. Figure 3 illustrates a study of our four databases with different sizes, and the average number of code lines generated for each product.All these data confirm the great productivity of our SPL.



	1	2	3	4
Pipes	363	1.551	5.781	11.361
Queues	1.413	5.793	21.393	41.993
Signals and Alerts	726	3102	11562	22722

Figure 3.*Productivity in big databases.*

5. Conclusions

This paper has showed the construction of a Software Product Line (SPL) using a generative programming approach. A new Domain Specific Language, called NCSL, has been developed to gather the domain variability. An adaptation of Exemplar Driven Development has been used to develop the SPL.

The Cost Model of this SPL based in SIMPLE has been presented andseveral sceneries have been analyzed, using this Cost Model.The cost of the SPL and the cost of the development of the products independently have been compared. This Cost Model presented has been applied to solve different study cases related to change notifications service in databases. In all these study cases the SPL cost is lower than the products development independently.

We have used another approach based in COPLIMO obtaining the productivity of the SPL. The conclusion with COPLIMO is the same: the great productivity of our SPL. This productivity increases with the database size. Even with very small databases, the SPL is much more productive than using traditional product development.

References

[1] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000. ISBN-13: 978-0201309775.

[2] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001. ISBN-13: 978-0201703320.

[3] Verlage, M.; Kiesgen*, T. Five years of product line engineering in a small company*. Proceedings of 27th International Conference on Software Engineering, 2005 (ICSE 2005), pp. 534-543. DOI: http://dx.doi.org/10.1145/1062455.1062551

[4] R Heradio Gil, J. F. Estívariz López, I. Abad Cardiel and J. A. Cerrada Somolinos. *Translation from Abstract Specifications to Executable Code via Exemplar Transformations*. V JornadassobreProgramación y Lenguajes (PROLE'05). Pages 185-191. 2005.

[5] Coz, J.R., Heradio, R., Cerrada, J.A. and Lopez, J.C. *A generative approach to improve the abstraction level to build applications based on the notification of changes in databases*. 10th International Conference on Enterprise Information Systems (ICEIS). Barcelona, Spain. 2008.

[6] Clements, Paul; McGregor, John; & Cohen, Sholom. *The Structured Intuitive Model for Product Line Economics (SIMPLE)* (CMU/SEI-2005-TR-003). Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/library/abstracts/reports/05tr003.cfm

[7] Böckle, Günter; Clements, Paul; McGregor, John D.; Muthig, Dirk; Schmid, Klaus. "*A Cost Model for Software Product Lines*", Fifth International Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 4-6, 2003.DOI: http://dx.doi.org/10.1007/978-3-540-24667-1_23

[8] Böckle, G.; Clements, P.; McGregor, J.D.; Muthig, D.;Schmid, K. "Calculating ROI for Software Product Lines", IEEE Software, Volume 21, Issue 3, May-June 2004, pages 23-31.DOI: http://dx.doi.org/10.1109/MS.2004.1293069

[9] B. Boehm et al. *Software Cost Estimation with COCOMO II*; Prentice Hall, 2000. ISBN-13: 978-0130266927.

[10] B. Boehm, A. W. Brown, R. Madachy and Y. Yang. *A software product line life cycle cost estimation model.* International Symposium on Empirical Software Engineering, 2004; 156-164.

[11] Oracle Documentation Library. http://www.oracle.com/technology/documentation/index.html

[12] Heradio, R. *Metodología de desarrollo de software basada en el paradigma generativo. Realización mediante la transformación de ejemplares*. Ph. D. Thesis, Departamento de Ingeniería de Software y Sistemas Informáticos de la UNED, España. 2007.

[13] *A Ruby implementation of EFL in RAA (Ruby Aplication Archive).*http://raa.ruby-lang.org/project/efl/DOI: http://dx.doi.org/10.1109/ISESE.2004.1334903

[14] J.R. Coz-Fernandez, R. Heradio-Gil, D. Fernandez-Amoros and J. Cerrada-Somolinos, "*A Domain Engineering Approach to Increase Productivity in the Development of a Service for Changes Notification of the Configuration Management Database*" Journal of Software Engineering and Applications, Vol. 6 No. 4, 2013, pp. 207-220. DOI: 10.4236/jsea.2013.64026

[15] Simon Adams. *ITIL V3 foundation handbook*. pp.7-11. TSO. 2009. ISBN: 978-0113311972.

[16] D. Fernández-Amorós, R. Heradio Gil and J. Cerrada Somolinos. *Inferring Information from Feature Diagrams to Product Line Economic Models*. ACM International Conference Proceeding Series; Vol. 446 archive. Proceedings of the 13th International Software Product Line Conference, 2009