Journal of Software Engineering and Applications, **2013, 6, 207-220**
http://dx.doi.org/10.4236/jsea.2013.64026 Published Online April 2013 (http://www.scirp.org/journal/jsea)

A Domain Engineering Approach to Increase Productivity in the Development of a Service for Changes Notification of the Configuration Management Database

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

Departamento de Ingeniería de Software y Sistemas Informáticos, Universidad Nacional de Educación a Distancia, Madrid, Spain.
Email: jrcozf@gmail.com, rheradio@issi.uned.es, david@lsi.uned.es, jcerrada@issi.uned.es

Received January 29th, 2013; revised March 2nd, 2013; accepted March 13th, 2013

ABSTRACT

This paper presents a domain engineering approach to build a software product line that supports the change notification service in a Configuration Management Database (CMDB) according to the Information Technology Infrastructure Library (ITIL) best practices. For the development of this product line, the proposed approach makes use of a construction of products methodology by analogy: this is a new notation which reports the variability of the products, obtaining metrics as important as the number of products and uses a language that enables, by means of the flexibilization of a product and the development of some generators, to build the rest of the product line. In addition the paper offers a standard for the analysis and design of the CMDB as well. Finally, the paper presents an economic model for the product line, where the profitability and productivity of the proposed solution are analyzed.

Keywords: ITIL; CMDB; Databases; Change Management; Configuration Management; Domain Engineering; Software Product Lines

1. Introduction

The importance of a proper operation and management, cost-effective and targeted to users and to the continuous improvement of services related to Information Technology and Communications (ICT), today, is crucial to ensure the survival of businesses and organizations. The standard Information Technology Infrastructure Library, in its version 3.0 [1], (ITIL) and the new concepts that support the operation of technological systems in organizations such as Configuration Management Database (hereinafter CMDB) or the Configuration Management Process are the core of technology management [2].

At present, there is a big challenge in organizations, using several frameworks, standards and regulations for the technology management. ITIL is one the most accepted framework in the organizations, including SME/SITU's companies [3]. The CMDB is established to record all stuffs, including assets, associated to IT departments in organizations. CMDB is widely used in many organizations [4,5].

This framework and these concepts are presented in this research project, which offers, within a domain engineering approach, a series of innovative tools and techniques for the development of a service for the notification of the changes into the CMDB. The research study includes standards for the lifecycle of a CMDB and a notation to document the variability of a software product line. This new notation allows for very important metrics for the development of SPLs as the number of products or commonality. It also presents a framework for development based on generative programming that allows us to get all the products that support the service for changes notification. Furthermore, the paper describes an economic study that allows us to obtain return on investment. Our framework is able to get thousands of products and the effort to carry it out is about a dozen products.

This article is structured as follows: Section 2 presents several concepts related to the article: ITIL best practices, the concept of CMDB and the notification of changes to databases and their relationship with the CMDB. The background investigation, as the variability in product

lines, generative programming and domain engineering are presented in Section 3. Section 4 describes the domain analysis and design. This section ends with a summary on the implementation of the domain. Section 5 presents an Economic Model for the solution, analyzing the productivity. Finally, in Section 6, the main conclusions are presented.

2. Preliminaries

2.1. About Information Technology Infrastructure Library

ITIL is today the "de facto" standard in most organizations for some processes related to the management of ICT services. Currently, the latest version 3.0 provides best practices that can serve as a benchmark for organizations to improve services.

First, about the design services including the management process of the availability; delivery, design and maintenance of service level agreements (SLA's) and catalog maintenance of ICT services provided by the organization [6]. Secondly, there are the operational services, which include event management processes, problems and incidents, Jan Van Bon [7]. Incident, according to ITIL, is any event that causes or may cause service interruption and problem is the underlying cause of one or more incidents. Event includes any situation that causes the service interruption.

Finally, there are transition services [8] that cover the process of change management, testing and validating systems, development, knowledge management and system configuration management.

2.2. About the Configuration Management Database

The Configuration Management, one of the main processes in the transition services, provides a logical model of the infrastructure or service through the identification, control, maintenance and verification of configuration items. Configuration Items (CI's) are the components of an infrastructure that are or will be under configuration control. CI's are unique and identifiable, are subject to change and can be managed. The CI has a set of standard attributes such as category, relationships, status and history. To know if something is a configuration item (CI) is necessary determining whether the organization has to manage to deliver an IT service. Another way to identify whether CI is an affirmative response test (USMC):

- Is it unique? (U)
- Is required to deliver an IT service? (S)
- Can you manage? (M)
- Do you have at least some features can change? (C)

The database containing all relevant data from each of the CI's, and details of the relationship between them is called CMDB (Configuration Management Database). This database is the only point of reference for all IT decisions and operations in the organization and provides visibility into the dependencies between business processes, users, applications and underlying IT infrastructure; residence and allows access to all CI. In general the CI's managed in the CMDB usually are at a very detailed level [9,10].

2.3. Changes Notification in Databases

The change notification service database is, very brief, the implementation of an observer of the changes in the database and a number of elements that subscribe to the observer for changes that occur therein. In the real world can give many examples of this type of scenario. Samples from [11]:

- A database supporting a university that offers courses. The system manager receives notifications from course subscriptions, depending on priorities and times that are set by the university administration.
- A control system for air navigation that contains electronic controls that inform pilots of changes in different aspects of navigation. The control system is supported by a database that receives information from different areas: weather, air bases, command flight and so on. When there is a change in certain parameters is necessary to alert pilots in order to support the decision making process during the flight and, in the same way, allow the aircraft's electronic systems run certain critical processes.
- A database is active from Monday to Friday, except weekends and holidays. Saturday's maintenance tasks are performed. It is necessary to have control over any change after hours or maintenance. For example, if a connection is made to the database during a holiday, this requires that a delegate is informed.

Each of these scenarios describes a situation in which messages are exchanged among multiple clients (nodes) in a distributed environment. Messages not only intercommunicate between the client and the server, but also between processes on the server itself.

If we focus on these scenarios in terms of messages, applications can be viewed as processes in which each step is caused by one or more messages, and results in one or more messages. Another way of saying this is that the messages are events that cause other events of message. Databases have several mechanisms that respond to these scenarios: pipes management, systems and signal alerts or advanced queue management, are the most common. For Pipes and Management Systems and Signal warning messages are transmitted in real time and there is no persistence or delayed notifications. In the case of advanced queue management, provides persistence and

ability to delayed notifications.

For the development of these features databases also have programming languages that extend the functionality of Structure Query Language (SQL) as PL-SQL or Transact SQL. These programming languages support all queries and data manipulation used in SQL, but includes new features such as handling variables, modular structures, exception handling, incorporating triggers, cursors advanced management and structures flow control and decision making.

Moreover, the databases have set libraries that allow extending the language by incorporating various mechanisms, including those for notification of changes. All these tools allow the development of services for notifycation of changes to databases; however, these products must be manually developed and customized as needed.

2.4. The Notification Change Service into a CMDB

In the case of a CMDB, the Notification of Changes has a very relevant as we shall discuss. Following the best practices offered by the ITIL framework, the CMDB is a federated database, which means that not all configuretion data must reside in one physical database [12].

The primary systems and data repositories are an authoritative source of information, while the CMDB becomes the reference where this information lives and how to access it. ITIL v3 now recognizes the importance of this approach and recommends as a fundamental part of the structure of a CMS (System Configuration Management). With federation, the basic data are stored in the CMDB, which is linked to other stores, with more detailed data. The federation allows access, through the CMDB, to all the configuration items (CI's).

From this source of information, which is the CMDB, whenever a change occurs in a configuration item (CI), this change may have an impact on the organization and other processes related to the CMDB. Section four of the paper, the analysis of the domain, shows the relationship between the CMDB and some of the main processes in the technological management of organizations. The processes will be supported by a number of information systems, such as asset management system or the system management of incidents and problems.

These systems may act as subscribers for notification of changes to the CMDB. This would allow the necessary changes in the CI are notified to the relevant systems. Section 4 mentions some of the potential subscribers to the CMDB.

3. Background in Existing Research

3.1. Software Product Lines and the Variability

The benefits of taking a Software Product Line (SPL) approach to develop similar software systems, in matter of quality, productivity and time-to-market, have been well documented by [13]. Key to the SPL approach is to exploit the commonalities and variabilities (*i.e.*, the differences) of the systems that belong to a SPL. At the moment, there is a wide variety of languages to document variability in SPLs.

The aim of building SPLs is to get an effective reuse of software. [14] summarizes the benefits of software reuse in the next points: increased dependability, reduced process risk, effective use of specialists, standards compliance, and accelerated development. On the eighties, some authors [15] estimated that the 60% of the software applications would be developed assembling reusable components. However, we are far from this forecast. For example, a report published on 2005 [16] revealed that the reuse level in 25 projects of the NASA with a size of 3000 - 112,000 lines of code was only the 32%.

According to many researchers, the development of single systems tends to produce an opportunistic reuse of software. On contrast, when multiple similar but distinct systems are produced collectively it is possible to reach a systematic reuse of software. Therefore, software engineering should move its focus from single systems to families of systems.

This approach was firstly proposed by D. Parnas [17] and nowadays is followed by different paradigms for building SPLs, such as generative programming [18], software factories [19] and software product line engineering [20], characterized to undertake the development of a set of products as a single and coherent development activity.

Clements, P. and Northrop, L. [13] define SPL as "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". This definition can be considered from two perspectives:

- Looking at the problem space, a SPL is a set of systems scoped to satisfy a given market. From this point of view, it is essential to identify the common and variable requirements of the systems.
- Looking at the solution space, a SPL is a set of systems sharing enough properties to be built from a core asset base.

From this second perspective, it is fundamental to decide how to implement the variability of the core assets; *i.e.*, the ability of the core assets to adapt to usages in the different product contexts that are within the SPL scope. Key to the SPL approach is to exploit the commonalities and variabilities (*i.e.*, the differences) of the systems that belong to a SPL. The terms feature and variation point/

variant are typically used to refer to the commonalities and variabilities in SPLs.

For convenience, in this paper we will talk about features in a general sense, according to the next definition: "a feature is a distinguishable characteristic of a concept that, from the perspective of the solution or the problem spaces, is relevant to some stakeholder of the concept".

Software reuse within an application domain depends on the discovery of the common elements of the domain products. There is a change of development oriented to a single software product development focused on several products that share some characteristics, forming a family.

There are two different processes: domain engineering, which focuses on the development of reusable components that will form the domain; and application engineering, that is oriented towards the development of individual products that satisfy a set of requirements and constraints expressed by a specific user. In our case, we opted for domain engineering.

One of the main objectives of the domain engineering process is to define the commonality and the variability of the SPL. This task is decomposed in:

- Domain Analysis, whose purpose is to scope the SPL domain, collect the relevant domain information and integrate it into a coherent model.
- Domain Design, whose goal is to develop architecture and a detailed design for the SPL and to devise a production plan.
- Domain Implementation, where the core assets are implemented.

The aim of the domain engineering process is to derive specific products by exploiting the variability of the SPL.

3.2. Generative Programming for Domain Engineering

Generative programming and model driven development consider the productivity gains in the performance of software needs to raise the level of abstraction of programming languages through the use of specifications and models.

Key to the success of these two paradigms is the automatic translation of models into executable code. For that machine translation is feasible the application domain models should decrease the variability between products that may be generated is significantly less than the commonalities.

One approach used is to apply transformations on the specification. Although there are specific languages for expressing transformations, require overcoming a significant learning curve and lack of some typical features of programming languages. Exemplar Driven Development (EDD) [21] is a Software Product Line method which takes advantage of the similarities among domain products to build them by analogy.

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. 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. The **Figure 1** illustrates a summary of EDD.

4. Overview of the Proposal

4.1. Domain Analysis

First, for a full domain analysis we begin by studying the main processes related to CMDB. About the *incident management process*, the CMDB provides a rich source of information to incident management [22].

Incident managers can quickly access CI status, determine impact by reviewing the relationships between CIs and the business applications they support and identify related CIs to restore service. For the *problem management process*, the CMDB gives a rich source of data for proactive problem management, accelerating and simplifying root-cause analysis and problem resolution. It can provide the immediate status of CIs affected by the problem. The CMDB can link incidents to problems, and help to visualize the problem and related CIs and their dependencies. It also can show the history of changes that may have caused the problem. CMDB data could automatically populate incident or problem records.

By respect the *configuration and assets management process*, the CMDB is key to configuration management, enabling the consistent, accurate, and cost-effective identification, control, status accounting, and verification of all CIs in the CMDB, Mohammad Sharifi, Masarat Ayat [23].

Release management process. CMDB information supports automated rollout across distributed locations by providing accurate, detailed information about hardware, software, and current configurations and their compatibility with changes that are incorporated in a release. The CMDB can keep version details for software, verifies tested configurations, and enables project scheduling.

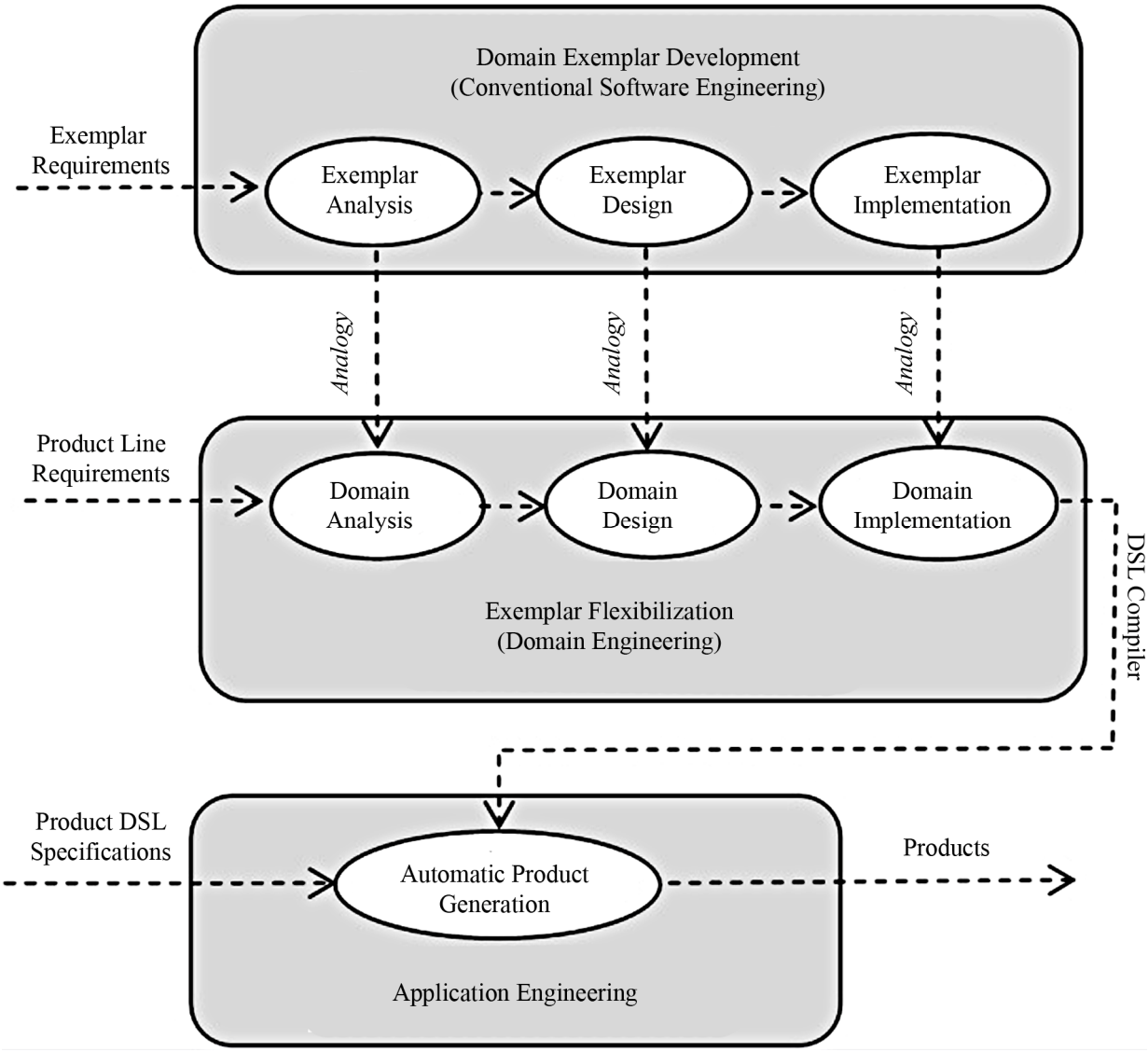


Figure 1. Exemplar driven development.

Service desk process. The CMDB can enable significant improvement in a range of service desk functions by providing detailed information about CIs related to service requests. Information about CI status, current configuration, configuration baseline, dependencies to other CIs and to business services, and planned changes all can help service desk managers meet service requests. The CMDB can also provide the data the service desk needs to notify users of outages and the status of problem resolution.

Service level management. The CMDB can allow end-to-end service level management, providing detailed data about CIs, their relationships to each other, and their relationships to the underlying IT infrastructure. It provides CI relationship data that links service level agreements (SLAs) to customers and to all related CIs that enable the service. It allows dynamic referencing of SLA components. Also, we can detail the interface with the *financial management process*. The CMDB provides information that is critical to the effective financial management of IT. It contains a complete list of CIs, from which easily can be produced expected maintenance costs and license fees, maintenance contracts, license renewal dates, and CI replacement costs.

Another process relevant is the *business continuity management*. The CMDB can store information about the information technology infrastructure components, their configurations and their dependencies to each other and key business processes. It also identifies the priority and the agreed-upon minimum level of business operation following a major service disruption.

The *availability management process* is as well an-

other interface for the CMDB. The CMDB can provide a central information repository that links availability, reliability, and maintainability of services to the underlying IT components. The CMDB can provide important business impact data, shows related components in an availability string, provides risk-analysis data, and helps isolate which CIs are the root cause of availability failures. The CMDB is essential for comprehensive *business capacity management process*. Information about CIs, their relationships with each other, and their relationships to business functions is necessary for automated capacity management.

About the *project management process*, the CMDB, along with a change and release management process, can provide the mechanism to identify, plan, track, update, and monitor the projects that create new CIs, modify CIs, or deploy instances of CIs. Having change and configuration management integrated into the project management lifecycle is critical to ensuring a smooth project-to-production transition and accurate CI status.

Other processes that can interact with a CMDB can be service *performance and quality management process* [24], *contract management*, *human resource management and training*. Finally, for the *audit*, *governance*, *compliance*, *and control processes*, the CMDB provides an essential repository of control-related data useful for both internal and external audits.

The Control Objectives for Information and related Technology framework, COBIT, recommends IT controls that can effectively leverage information from a CMDB. Each time a change occurs in a CMDB all these processes may be affected and in this scenario the changes notification service becomes a powerful and useful tool.

Another issue discussed in our research is the potential targets to subscribe to changes in the CMDB. Some of them are: the Asset Management System, Incident/Problem/Change Management Systems, Event Monitoring Systems, Directory Systems (user details, locations, association of hardware location to user consumption, etc.), Definitive Media Library (physical license store, archive of all in-house developed code, licenses, master copy of commercial off-the-shelf (COTS), software packages, etc.), Document Management System or Human Resource and Financial systems.

For clarity, the following is a small example. Service Desk is the single point of contact for the users who need help for running their IT systems. Customers contact the service desk for various purposes such as information, configuration change, problems, etc.

Customers can report problems using an Incident Management System (IMS). Service Desk tries to satisfy the customer requests to facilitate the restoration of normal operational service with minimal business impact. The IMS may be a subscriber to the service. This service might report, for example, when a web server is down. Thus, if the client asks for internet access, service desk can report the problem to the customers.

4.2. Domain Requirements

Whatever the solution chosen for the construction of the CMDB (integrated multiple data repositories, one single centralized data repository federated data repositories or with one central data repository), within our domain is necessary to consider a multitude of requirements that must be analyzed. The purpose of this section is not to expose all of them, only give an overview of the most relevant:

- Time management.
- Subscribers management.
- Granularity.
- Priorities management.
- Navigational management.
- Searches management.
- Visibility management.
- Queues management.

The initial set of requirements is related *to time management*. This set of requirements includes, in turn, several components. The first component is management of retentions. In the case of using a mechanism such as queue management, and reported the messages remain in the queue during the time that is determined by the retention. Another aspect that includes time management is the management of delays. Delays down the length of time since the CMDB know a change until it is notified from the CMDB to the corresponding subscribers.

Another component of time management is the management of timeouts. The timeouts are the time elapsed since the CMDB know a change until it is considered expired and is no longer notifies to the subscribers. The last key aspect of time management is to manage the waits. This requirement is related to mechanisms that provide persistence, and queue management, and refers to the waiting time of messages to be placed in the queue of notifications.

The second set of requirements is the *management of subscribers* to the notification mechanism. There may be a single subscriber as a messaging system, or there may be multiple subscribers to be notified about changes in the CMDB. As discussed in the section on the change notification service into a CMDB, in our proposal, we analyzed a set of candidate systems to be the mechanism for notifying subscribers of changes to the CMDB and included systems such as the Asset Management System, Incident/Problem/Change Management Systems and others. It is necessary to identify all such systems will be eligible for this service for our organization.

The third group of requirements is the *granularity of the solution.* We may find different solutions as a coarse granularity, where change management is done at database level. That means that only general operations on the database will be notified. For example, when starting the CMDB, when there is a change in it, when the CMDB falls and others. We can meet with a medium grain granularity where change notification is made at the entity level. For example, when there is a change of software in a financial system notifies to the helpdesk system. And finally, we can have a fine-grained solution where change management can be performed at the level of data of the CMDB. Any data changed could be notified.

The fourth group of requirements to be analyzed is the *management of priorities*. Different priorities among the messages to be notified can be set. Events with more level of priority could be notified first and events with less level could be notified last. For example, a change in a software component of the financial system can be considered very critical and will be notified first, while a change of system hardware purchasing department can be considered less critical and will be notified last.

A fifth group of *requirements relates to the navigation*. This group establishes how to navigate through the messages in the case of choosing a queuing mechanism. When scrolling through the queue, it can set this requirement as "first message", so it always go to the first message from the queue, or "next message", to move to the next or "next transaction", to move to the message that corresponds to the next transaction.

Another group of requirements is the *management of searches*. This group of requirements sets the "standard" search messages in queues notification. For example, we have in our organization a CMDB that contains all configuration items related to the system of the corporate website. This system is a subscriber to the service changes notification. The corporate website want to be notified only those software changes the text in the comments displayed the word "Web".

There are other more detailed sets of requirements as the *management visibility*. This set of requirements determines the manner in which the messages queued or dequeued. Overall, we have a transactional or immediate type. In the first case, until the transaction is not completed, the message is not queued/dequeued. In the second, is queued/dequeued before the transaction occurs.

Another group of requirements is the *management of the queues*. These are related to how to dequeue messages. There is a navigational model, where the notifications do not affect the messages, a lock mode, where messages that are reported are blocked and a deleted mode, where messages that will notify subscribers are removed from the queue.

4.3. Domain Design and Documenting Variability

Since the first Feature Diagrams notation was proposed by the FODA methodology in 1990, a number of extensions and alternative languages have been devised to model variability in families of related systems [25].

Several authors propose the Varied Feature Diagram+ (VFD+) as the language for documenting variability [26]. VFD+ introduces unnecessary complexity for automatic variability management of diagrams. For instance, as recognized by the authors themselves in [27] diagram satisfiability can be faster evaluated in trees.

To overcome these objections, we propose the Neutral Feature Tree Easy (NFTE) notation, derived from NFT [28]. NFTE is an extension of NFT. NFT has the same expressiveness, embed ability and succinctness as VFD+. In fact, NFT is a VFD+ subset where diagrams are restricted to be trees. We propose to use NFTE. NFTE uses more comfortable serialization syntax of NFT, where nodes of the diagram are specified as:

node("*node_name*",[*list_of_children*],*low*,*high*)

where low and high are 1 by default and *list_of_children* is optional for leaf nodes. Constraints are written in Conjunctive Normal Form as:

constraint("*node_*1"|*neg*("*node_*1"),..,
"*node_n*"|*neg*("*node_n*"))

Comments may be written by starting a line with the # symbol. A simple example is shown on **Figure 2** (Signals and Alerts Control Mechanism, SACM), which includes three requirements (Waits, Granularity and Subscribers) and one restriction: With No Waits (represented by DSE) is only possible Coarse Grain (represented by GG):

node("*SACM*",["*Subscribers*", "*Granularity*", "*Waits*"],3,3)
node("*Subscribers*",["*SU*", "*SM*"],1,1)
node("*Granularity*",["*GG*", "*GM*", "*GF*"],1,1)
node("*Waits*",["*DSE*", "*DECV*"],1,1)
constraint("*DSE => GG*")

This is a simplified example of one of the mechanisms for managing notifications with a handful of requirements.

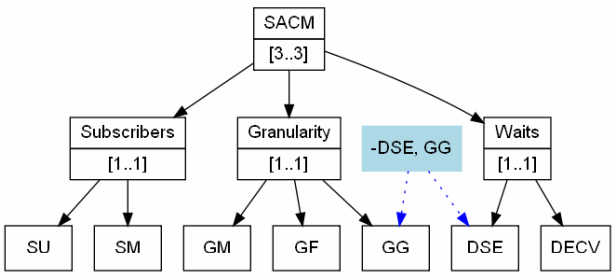


Figure 2. Graphical representation of NFTE.

When handling a large set of requirements, graphical diagrams become too unwieldy. In some of our examples we have to handle more than 100 requirements.

To avoid these problems and have a graphical representation, we have implemented an Interface Development Environment (IDE) that allow the transformation from NFTE notation diagrams to input format of graphical tools that allow us to view and manipulate the diagrams in a much automated form.

In addition, this IDE offers support for some of the features we need as the total number of products, the homogeneity of the SPL or the degree of reuse (commonality).

Figures 3 and **4** show some graphs obtained with data received from our IDE for the previous example: the number of products of every requirement and their commonality.

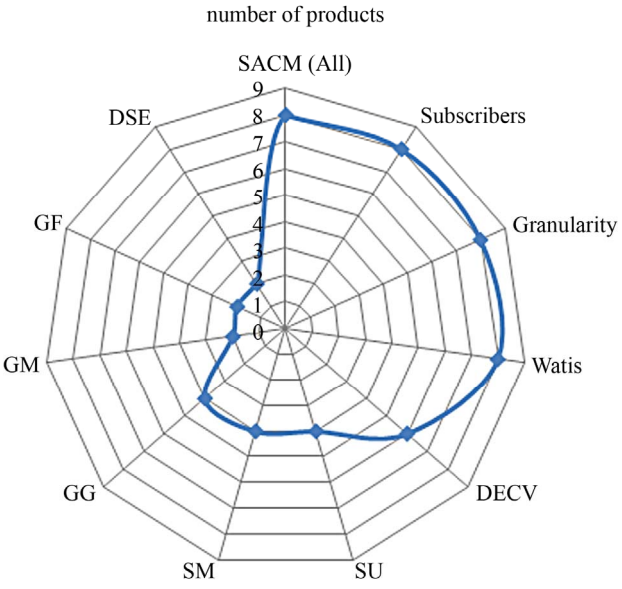


Figure 3. Number of products in our sample.

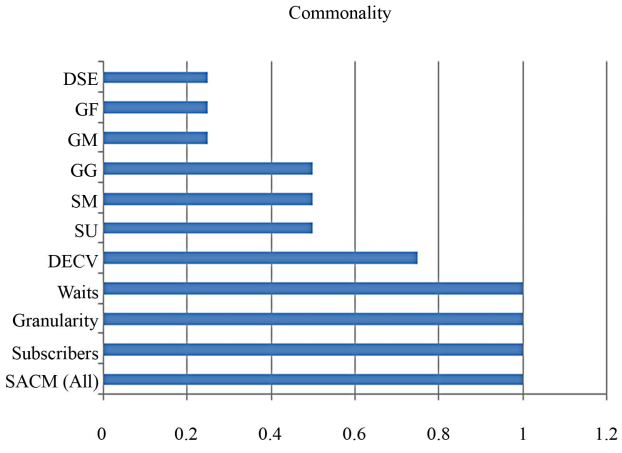


Figure 4. Commonality in our sample.

4.4. CMDB Standardization

When considering the development of product families one of the key issues is the standardization of design and analysis. To this end, we propose the standardization of the CMDB. First, we made a proposal for standardization of the major components of life cycle of the CMDB:

- The categorization of configuration items.
- Conceptual Design.
- Logical Design.
- Physical Design.

About the *categorization* in our proposal the IT components should fall into predefined, standardized categories, each one containing like-CIs. In the infrastructure configuration structure, categories encompass similar CIs that are then further detailed through parent-child relationships, where children are specializations of their parent. Categories can help how much detail is required in each group of similar CIs. For example, consider the category of hardware. The next level down could contain generic categories of hardware, such as servers, workstations, and routers. **Table 1** provides examples starting point for each component categories. Secondly, we proceed to standardize the attributes of each configuration item. In **Table 2** it can be seen a selection of attributes for each of the categorized items.

According to our proposed, *conceptual model* of a CMDB is used in the early stages of the life cycle, and it identifies the main entities and attributes, candidate keys, domains and the CMDB, which may be multiple, recursive generalizations, aggregations and others. The main objective of such models is to establish the scope of information to manage the CMDB. Will be accompanied by a description of the business rules of the CMDB and should not be taken into account the needs of existing technology, or other restrictions, as established by the methodologies for this purpose.

This standard is based on writing the names of data elements (attributes, columns) using three basic terms: an entity (class), an attribute (property) and the representation, in line with other standards as ISO 11179, Information Technology-Specification and Standardization of Data Elements. Some of these rules will be extended to logical and physical models of the CMDB and our proposal are identified as common standards. As an example we present some of them:

- All objects in the CMDB (entities, attributes, relationships, etc.) must have a dossier containing: name, definition, comments and, if necessary, units of measurement.
- It will use a restricted character set, to ensure portability in environments where the character is reduced. The proposed set of characters allowed is based on using those admitted to name tags in an XML docu-

Table 1. CMDB categories.

Infrastructure CIs and categories	Examples
Application Software	Code, language, build, compiler.
Support Software	Operating system, build image version.
Hardware	Manufacturer, Serial number, mac address, IP address, firmware.
Data	Client, Location ID.
Service Unit	Bandwidth threshold, cost per month.
Process	Name, owner.
Standard	Date, name.
Documentation	Version, author, editor.
Facilities	Location, contact.

Table 2. Configuration items attributes.

Infrastructure CIs and Categories	Attributes
Application Software	Presentation layer module, Presentation logic layer, Business logic layer, Data access layer module.
Support Software	Operating system, virtual server, antivirus, backup sw, server base image, workstation base image.
Hardware	Desktop, Laptop, printers, network devices (router).
Data	Client data, location.
Service Unit	Network, desktop units.
Process	Service request, work instruction, procedure.
Standard	Policy, procurement, security standard.
Documentation	Service blueprint, service agreement, support documentation.
Facilities	Data center, remote office.

ment, considering it a de facto standard platform-independent.

- The table of characters allowed in the CMDB will be: Numeric: 0 - 9 ASCII codes (48 - 57), alphabetical: AZ, az ASCII codes (65 - 90) and (97 - 122) or underline, ASCII code (95).
- The character set for definitions and comments contained in the CMDB must be compatible with ISO-8859 standard.
- Establishing a set of very detailed rules on abbreviations and acronyms.
- Regarding the name of each object is defined a series of criteria relating to maximum size, the number of uses, separation of words, prepositions, articles and conjunctions, the compound names to the use of verbs, the use of proper names or organizations, standards, systems, interfaces and others.
- Should apply the principle of uniqueness of names within the same level of abstraction (in the case of conceptual models within the same model and the same applied to the rest).
- While the CMDB model consists of several submodels, are not permitted relationships of an entity (or table in the case of other models) with entities (or tables or table in the case of other models) that are not included in the same model, to avoid the complexity of the changes control. This includes the attributes necessary to identify the model and criteria for change control and versioning.
- It identifies a set of criteria for the domains, defined as a data type defined from one of the basic data types of the methodology, technique or tool used, which has a functional meaning in the context that applies to one or more attributes of the model.
- Subsequently, we propose to define a set of rules specific to the conceptual model of the CMDB. Some samples of these rules are:
- As far as possible primary keys must be identified, at least for major institutions.
- Include the attributes considered most significant functional context.
- Use, where necessary, hierarchies of super types and subtypes to represent certain real-world structures (generalization, specialization, categorization, inheriance, etc.).
- IDEF1X [29] use is recommended as notation for the conceptual model of the CMDB.
- Establishing a series of very specific rules and look details for the appointment of entities, attributes and relationships.

The *logical data model* of the CMDB is obtained from solving the conceptual model complex relationships, eliminating redundancies and ambiguities, identifying dependency relationships, completing entities and attributes, iden-

tifying the keys of each entity and specifying cardinality. Associated with the logic model should estimate the growth of the institution, the type and frequency of access as well as those features relating to security, confidentiality, availability, etc., considered relevant. This type of model is linked to technology, relational databases. In our proposal, the following rules apply to the logical model of the CMDB; also will include general rules established earlier:

- The logical model is derived from the conceptual model.
- Should include all entities and all the attributes, not only the most significant.
- Relations are not allowed "many to many" must be resolved with partners involved.
- The foreign keys (FK's) must be migrated and identified.
- The alternatives keys must be defined and identified.
- When necessary, recommend the use of numerical surrogate keys (or auto number), which must be defined and identified.
- Not required to solve the hierarchy of super types and subtypes. It is recommended not to do at this level of abstraction.
- IDEF1X notation must be used.
- It is mandatory that the model is, at least in 2nd normal form (FN), and highly recommended, in 3rd NF.
- Establishing a series of very specific rules and look details for the appointment of tables, columns, relationships, key partners and migrated keys.

The *physical model* of the CMDB is obtained from the normalized logical data model, analyzing the technical characteristics of the database manager to use, estimating volumes and setting rates and other operator-dependent as sample blocking, compression data or clusters. During the implementation of a system will be necessary to implement the objects of the logical data model, adapting to the limitations of a particular software system (relational database management system, operating system, language, etc.). It also tends to cause a de-normalization process, because of which there are new objects of data models physical level. Finally it is the need to identify specific objects of physical models such as indexes, sequences, constraints and others.

One of the usual limitations on the way to the physical data model refers to the maximum size of the names of the tables, columns, fields, etc. This shortening of names is also usually due to practical reasons that make writing code, but should not be misused, as it makes it more difficult to understand. Therefore, the designer will be forced to shorten the names of data objects of the logical model, usually through the use of abbreviations. In our proposal, the following rules apply to the physical model of the CMDB; also will include general rules established earlier.

- The physical model is derived from the logical model.
- Alternatives should include keys and inverse input (inversion entries).
- The hierarchy of super types and subtypes should be resolved having applied the necessary changes.
- IDEF1X notation must be used.
- Establishing a series of very specific rules and look details for the appointment of tables, columns, relationships, triggers, names of data objects, instances, primary key constraints, unique keys and shortening the logical names.

4.5. Domain Implementation

When developing our service for notification of changes we can find two types of situations: our organization has a CMDB or is necessary to develop a CMDB from scratch. In the case that we had to develop a CMDB, we recommend to follow our guidelines and make use of Common Information Model, CIM [30]. This model is a de facto standard to represent and organize all the information management of a technological environment in an organization.

Once developed the CMDB, our initiative is designed to take advantage of the proximity of the products that can be generated within a domain, suggesting that the implementation of changes not only on the specification, but on any of the products. The translator is defined as a program that takes a product previously developed domain, which is identified as exemplar, and transforms it to fit a specification. That is, in order to take advantage of the proximity of the products that can be generated within a domain, it is proposed that the changes to be implemented on any of the products. We use an adaptation of Exemplar Driven Development (EDD) [21], where the NFTE diagrams are built specifying the user features and using the necessary information from the database.

This database information is contained in metatables and it is obtained automatically. Once the domain specific language exist (in this case, the NFTE diagrams), the Domain Specific Compiler (DSC) for this language is implemented. A summary of the EDD adaptation is illustrated in the **Figure 5**.

To develop the products is used the Exemplar Flexibilization Language (EFL). EFL is an external flexibilization technique that supports noninvasive exemplar transformations and crosscutting flexibilization. It is applicable to whatever kind of software artifact and provides an efficient generative variant construction. EFL is used to build the DSC that deal with the specification variability and also with the implementation variability in our domain case study.

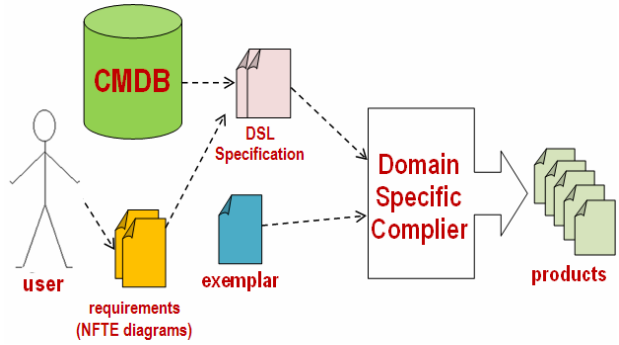


Figure 5. Adaptation of EDD to build the CMDB notification change mechanism.

Figure 6 illustrates a typical DSC written in EFL, made of an analyzer which takes as input a specification, and a generator which is responsible for generating the new product. The generator as showed in the figure could be made of other generators. EFL is currently implemented as a library of the Ruby object oriented language.

This implementation is distributed as Lesser GNU General Public License and it is available in different repositories as Ruby Forge or RAA (Ruby Application Archive).

The most important part is that generators are responsible for analyzing the exemplar and adapt it in order to generate the new product according to the given specification. Decomposing the generators in other generators helps to implement the flexibilization, since each generator will dealt with different artifacts fragments in which changes has to be done. On these artifacts fragments different specific analysis capabilities are required. Modularization of changes and traceability are also reasons that drive us to decompose in different generators.

Generators are also responsible for detecting dependencies and inconsistencies in the configuration model. This capability, in the SPL presented in this paper, is considered essential because the user might have selected wrong requirement or the requirements could contain among them incompatibilities, as combinations not allowed. This could drive to an invalid product for the SPL. In case of miss configurations generators provide a detailed report about the incompatible features. The user can use this report to review the selected features. Finally, generators can analyze the internal elements of the database to obtain all the necessary information of the domain.

5. Economic Study

Our mathematical model for calculating the benefits provided by our proposal is based on the standard COPLIMO [31]. COPLIMO is a COCOMOII [32] extension. According to this standard and having to:

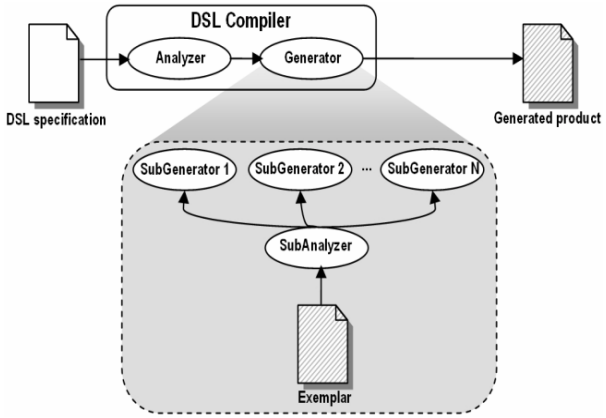


Figure 6. Domain Specific Compiler in EFL.

- 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 first Equation (1) of our economic model:

$$\mathrm{PLS}(\mathrm{N}) = \mathrm{PMNR}(\mathrm{N}) - \mathrm{PMR}(\mathrm{N}) \tag{1}$$

where PMNR(N) is estimated using the standard COCOMO II:

$$\mathrm{PM} = A \times \mathrm{Size}^{E} \times \prod_{i=1}^{n} \mathrm{EM}_i \tag{2}$$

- 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.
- EM_i are Effort Multipliers (required software reliability, database size, product complexity, required reusability).

Then, we have:

$$\mathrm{PMNR}(\mathrm{N}) = N \times A \times \mathrm{Size}^{E} \times \prod_{i=1}^{n} \mathrm{EM}_i \tag{3}$$

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

$$\mathrm{PMR}(1) = \mathrm{PMNR}(1) \times \left(\mathrm{PFRACRCWR} \times \left(\mathrm{RFRAC} + \mathrm{AFRAC}\right)\right) \tag{4}$$

$$\mathrm{PMR}(\mathrm{N}) = \mathrm{PMR}(1) + (\mathrm{N}-1) \times \mathrm{PMNR}(1) \times \left(\mathrm{PFRAC} + \mathrm{RFRAC} \times \mathrm{AA}/100 + \mathrm{AFRAC} \times \mathrm{AAM}\right) \tag{5}$$

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 sw reusable across the SPL. If RUSE is the development for reuse, DOCU is the degree of documentation and RELY is the required software Reliability:

$$\mathrm{RWCR} = \mathrm{RUSE} \times \mathrm{DOCU} \times \mathrm{RELY} \tag{6}$$

- A (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. **Table 3** shows the values.

For our Software Product Line (SPL) we have:

$$\mathrm{PMNR}(\mathrm{N}) = N \times A \times \mathrm{esize}^{E} \times \prod_{i=1}^{n} \mathrm{eEM}_i \tag{7}$$

where esize is the size of the exemplar and eEM are the Efforts Multipliers of the exemplar. The "scaling parameter" is:

$$E = B + 0.01 \times \prod_{i=1}^{n} \mathrm{SF}_i a \tag{8}$$

where B is the "scaling base exponent for the effort" and SF_i are the scale factors: precedentedness, development flexibility, architecture, risk resolution, team cohesion and process maturity.

The cost for building n products in our SPL is:

$$\mathrm{PMR}(1) = \mathrm{PMNR}(1) + \sum_{i=1}^{G} \left\{ \mathrm{A} \times \mathrm{gsize}^{E} \times \prod_{i=1}^{n} \mathrm{gEM}_i \right\} \tag{9}$$

$$\mathrm{PMR}(\mathrm{N}) = \mathrm{PMR}(1) + (\mathrm{N}-1) \times \frac{\mathrm{AA}}{100 \times \left\{ \sum_{i=1}^{G} \left\{ \mathrm{A} \times \mathrm{gsize}^{E} \times \prod_{i=1}^{n} \mathrm{gEM}_i \right\} \right\}} \tag{10}$$

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

$$\mathrm{ROI}(\mathrm{N}) = \frac{\text{cost savings}}{\text{cost investiment}} = \frac{\mathrm{PLS}(\mathrm{N})}{\left|\mathrm{PLS}(1)\right|} \tag{11}$$

We get all the parameters of our CMDB Notification Service Economic Model (CMDB NS EM). Some of them are listed in the **Table 4**.

Substituting all parameters into the formulas and where N is the number of products in our SPL we obtain that the number N of products necessary for our product line has benefits is:

$$\mathbf{ROI}(\mathbf{N}) => 0 => N > 11$$

That is, with only 11 products, our SPL will be productive. The number of products obtained with our SPL varies depending on the requirements, the number of subscribers, requirements and the size of the CMDB, mainly. In reference to the size of the CMDB, in the case of opting for a fine-grained granularity, the notification changes service will be able to report changes at the entity level of the CMDB. The scope of our SPL and the number of products will grow with the number of entities.

Table 3. Assessment and assimilation.

AA increment	Description
0	None
2	Basic module search and documentation
4	Some module Test and Evaluation (T & E), documentation
6	Considerable module Test and Evaluation (T & E), documentation
8	Extensive module Test and Evaluation (T & E), documentation

Table 4. Parameters for our model.

Parameter	Description	Value
A	Effort coefficient	2.94
B	Scaling base-exponent for effort	0.91
$\sum \mathrm{SF}_j$ (gen)	Sum of all Scale Factors for the Generators	6.32
$\prod \mathrm{Em}_j$ (gen)	Product of 17 Effort Multipliers for the Generators	2.33
E (gen)	Scaling exponent for effort (Generators)	0.97
Size (ex)	Size of the Exemplar in PM (person/months)	0.25
Size (gen)	Size of the Generators in PM (person/months)	2.00
AA	Assessment and Assimilation	4.00

Using the algorithms described in [28] we obtain the relationship between the number of entities and the number of products. As we can see, the number of possible products of our SPL grows exponentially with the number of entities considered in our service, as shown in **Figure 7**. In reference to the subscribers, we obtain the relationship between the number of subscribers and the number of products.

As we can see, the number of possible products of our SPL grows exponentially with the number of subscribers considered in our service, as shown in **Figure 8**.

In our analysis we identified a number of requirements and features and managing a case of fine-grained with a CMDB with 8 entities and 5 subscribers, we get the number of products shown in **Figure 9**.

6. Conclusions

We have presented a framework for building a product line that enables us to implement the change notification service in a CMDB (Configuration Management Database), according to ITIL (Technology Infrastructure Library) best practices. Both ITIL and CMDB as the

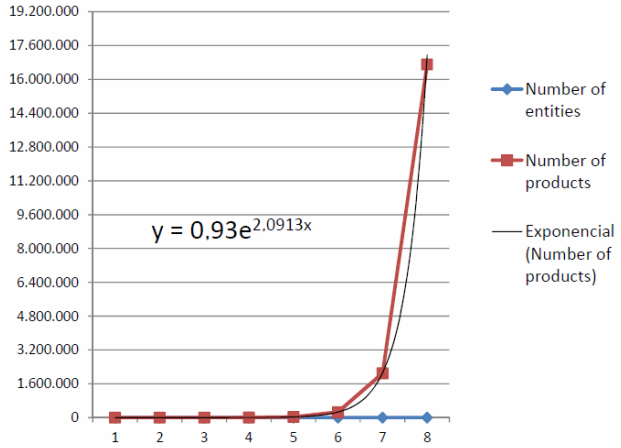


Figure 7. Number of Products and Entities.

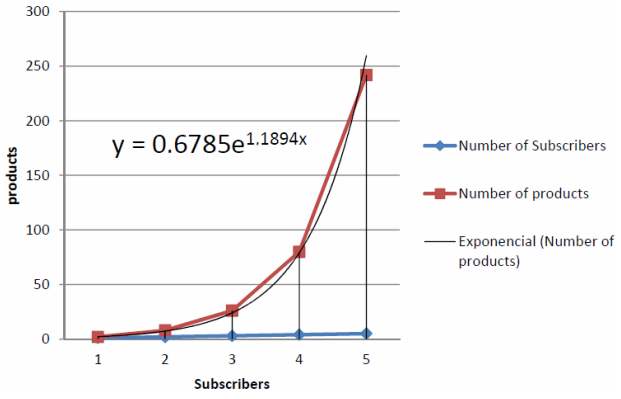
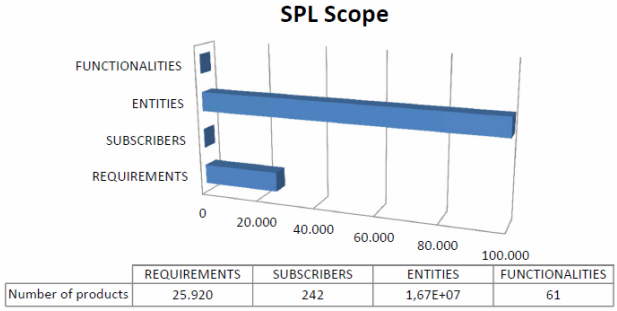


Figure 8. Number of Products and Subscribers.



	REQUIREMENTS	SUBSCRIBERS	ENTITIES	FUNCTIONALITIES
Number of products	25.920	242	1,67E+07	61

Figure 9. Number of Products.

mechanism for notification of changes to the multiple and potential subscribers are crucial elements in the current workings of all modern IT organizations.

The framework presented, located under the umbrella of the engineering domain, makes use of innovative techniques and tools such as EDD (Exemplar Driven Development) methodology aimed at developing product lines built by the analogy of a product, NFTE (Neutral Feature Tree Easy) as a notation for documentation of the variability of the product line, EFL (Exemplar Flexibilization Language) that allows us, through the construction of generators, from the specification of a product of the SPL, to get the rest of the products, or the calculation of the number of products by SPL using algorithms from the NFTE notations.

In addition, the work presented is offered for a CMDB standard. This standardization, really useful in developing our product line, can be extended and generalized to any standard of design and specification of a CMDB. Finally, we presented an economic model to our line of products based on COPLIMO, where we have obtained, as a most important conclusion: the great profitability and productivity of our product line. Our framework is able to get thousands of products and the effort to carry it out is about a dozen products.

REFERENCES

[1] S. Adams, "ITIL V3 Foundation Handbook," TSO, 2009, pp. 7-11.

[2] G. Blokdijk, "CMDB and Configuration Management Process, Software Tools Creation and Maintenance, Planning, Implementation Guide," Lulu.com, 2009, pp. 63-68.

[3] M. Ayat, M. Sharifi, S. Ibrahim and S. Sahibudin, "CMDB Implementation Approaches and Considerations in SME/SITU's Companies," *3rd Asia International Conference on Modeling & Simulation*, Bali, 25-29 May 2009, pp. 381-385. doi:10.1109/AMS.2009.113

[4] M. X. Atherton, "Deploying CMDB Technology Pragmatism and Realism Will Deliver the Benefits," Freeform Dynamics Ltd., 2009. http://www.freeformdynamics.com/

[5] R. A. Steinberg, "Implementing ITIL: Adapting Your IT Organization to the Coming Revolution in IT Service Management," Trafford Publishing, Trafford, 2005, pp. 50-54.

[6] J. Van Bon, "Service Design. Service Design Based on ITIL V3: A Management Guide Best Practice," Van Haren Publishing, 2009. http://www.vanharen.net/

[7] J. Van Bon, "Service Operation. Service Operation Based on ITIL V3: A Management Guide Best Practice," Van Haren Publishing, 2008. http://www.vanharen.net/

[8] J. Van Bon and M. Pieper, "Service Transition Based on ITIL V3: A Management Guide Best Practice," Mike Pieper, 2008. http://www.vanharen.net/

[9] R. J. Colville, "Gartner RAS Core Research Note G00137125," Gartner on CMDB, 2006.

[10] M. Corp, "Microsoft Operations Framework: Capacity Management Service Management Function," 2005.

[11] J. R. Coz, R. Heradio, J. A. Cerrada and J. C. Lopez, "A Generative Approach to Improve the Abstraction Level to Build Applications Based on the Notification of Changes in Databases," 10*th International Conference on Enterprise Information Systems*, Barcelona, 2008.

[12] D. Clark, *et al*., "The Federated CMDB Vision: A Joint White Paper from BMC," Technical Report, CA, Fujitsu, HP, IBM, and Microsoft, Version 1.0., 2007.

[13] P. Clements and L. Northrop, "Software Product Lines: Practices and Patterns," Addison-Wesley, Boston, 2001.

[14] I. Sommerville, "Software Engineering," 9th Edition, Addison-Wesley, 2010.

[15] L. G. Lanergan and C. A. Grasso, "Software Engineering with Reusable Designs and Code," *IEEE Transaction on Software Engineering*, Vol. 10, No. 5, 1984, pp. 498-501. doi:10.1109/TSE.1984.5010273

[16] R.W. Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems," *IEEE Transactions on Software Engineering*, Vol. 31, No. 6, 2005, pp. 495-510. doi:10.1109/TSE.2005.69

[17] D. Parnas, "On the Design and Development of Program Families," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, 1976, pp. 1-9. doi:10.1109/TSE.1976.233797

[18] K. Czarnecki and U. Eisenecker, "Generative Programming: Methods, Tools, and Applications," Addison-Wesley, 2000.

[19] J. Greenfield and K. Short, "Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools," 1st Edition, Wiley, 2004.

[20] K. Pohl, G. Böckle and F. Linden, "Software Product Line Engineering: Foundations, Principles and Techniques," Springer, 2005.

[21] R. H. Gil, J. F. E. López, I. A. Cardiel and J. A. C. Somolinos, "Translation from Abstract Specifications to Executable Code via Exemplar Transformations," V Jornadas sobre Programación y Lenguajes (PROLE'05), 2005, pp. 185-191.

[22] R. Gupta, J. H. Prasad and M. Mohanla, "Automating ITSM Incident Management Process," *International Conference on Autonomic Computing*, Chicago, 2-6 June 2008, pp. 141-150. doi:10.1109/ICAC.2008.22

[23] M. Sharifi, M. Ayat, S. Ibrahim and S. Sahibudin, "A Novel ITSM-based Implementation Method to Maintain Software Assets in Order to Sustain Organizational Activities," 3*rd UKSim European Symposium on Computer Modeling and Simulation*, Athens, 25-27 November 2009, pp. 274-280. doi:10.1109/EMS.2009.73

[24] M. Sharifi, M. Ayat and S. Sahibudin, "Implementing ITIL-Based CMDB in the Organizations to Minimize or Remove Service Quality Gaps," 2*nd Asia International Conference on Modeling & Simulation*, Kuala Lumpur, 13-15 May 2008, pp.734-737. doi:10.1109/AMS.2008.144

[25] P. Schobbens, P. Heymans and J. Trigaux, "Feature Diagrams: A Survey and a Formal Semantics," 14*th IEEE International Conference on Requirements Engineering*, Minneapolis, 11-15 September 2006, pp. 139-148. doi:10.1109/RE.2006.23

[26] P. Heymans, P. Schobbens, J. Trigaux, Y. Bontemps, R. Matulevicius and A. Classen, "Evaluating Formal Properties of Feature Diagram Languages," *Software*, *IET*, Vol. 2, No. 3, 2008, pp. 281-302. doi:10.1049/iet-sen:20070055

[27] P. Schobbens, P. Heymans, J. Trigaux and Y. Bontemps, "Generic Semantics of Feature Diagrams," *Computer Networks*, Vol. 51, No. 2, 2007, pp. 456-479. doi:10.1016/j.comnet.2006.08.008

[28] D. Fernández-Amorós, R. H. Gil and J. C. Somolinos, "Inferring Information from Feature Diagrams to Product Line Economic Models," *Proceedings of the* 13*th International Software Product Line Conference*, Vol. 446, 2009, pp. 41-50.

[29] T. Bruce, "Designing Quality Databases with IDEF1X Information Models," Dorset House Publishing Company, 1991.

[30] "Common Information Model (CIM)," Version 2.26.0., DMTF Application Working Group, 2011.

[31] B. Boehm, *et al*., "Software Cost Estimation with COCOMO II," Prentice Hall, Upper Saddle River, 2000.

[32] 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*, 19-20 August 2004, pp. 156-164. doi:10.1109/ISESE.2004.1334903

