PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

LABORATORIO WEB PARA ESTIMAR LA DEMANDA DE CURSOS EN LA ESCUELA DE INGENIERÍA EN EL PRIMER CICLO FORMATIVO

ERWIN ANDRÉS AGÜERO MEZA

Memoria para optar al grado de
Ingenierio Civil Industrial, con Diploma en Ingeniería en Computación

Profesor Supervisor:
JAIME NAVÓN COHEN

Santiago de Chile, Agosto 2017

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

LABORATORIO WEB PARA ESTIMAR LA DEMANDA DE CURSOS EN LA ESCUELA DE INGENIERÍA EN EL PRIMER CICLO FORMATIVO

ERWIN ANDRÉS AGÜERO MEZA

Miembros del Comité:
JAIME NAVÓN COHEN
YADRAN ETOROVIC SOLANO
DANIEL OLIVARES QUERO

Memoria para optar al grado de
Ingenierio Civil Industrial, con Diploma en Ingeniería en Computación

Santiago de Chile, Agosto 2017

A todos aquellos que contribuyeron
en mi paso por la universidad :)

AGRADECIMIENTOS

Write in a sober style your acknowledgements to those persons that contributed to the development and preparation of your thesis.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABSTRACT

The aim of this project is create a platform which allows to estimate and predict the number of students will take a course in The Engineering School in the next academic period. This need borns for the implementation of the new acamedic plan created in 2013, this program is divided into 2 formative phases: Bachelor of Engineering, during the first 4 years, and follows for an UC Professional Title or another academic degree. The flexibility that allows this structure has as consecuence that one specific course can be founded in the acamedic path of very diverse students, it brings a really high variability in the number of students whom take the course.

In order to manage the courses demand was created a web plataform using the framework Ruby On Rails and R ,the language and environment for statistical computing and graphics. These technologies allow to face in a modular way the challenges to create predictives models and manage the information of them through the web platform. In the the requirements gathering process was used the methodology User Stories and for the design of predictive models was choosen Cross Industry Standard Process for Data Mining (CRISP-DM). In general, the web platform manage all the information used by the predictive models and allow load new data, update and/or delete the current information, at the same time, it allows to operate the predictive models for each course and visualize the results obtained.

To conclude, the results of the predictions are presented based on the data provided by the *Dirección de Pregrado*, as well as a list of solutions to the problems that exits for the lack of data given that there is still no generation of students who have completed the first cycle.

RESUMEN

El objetivo del presente trabajo es crear una plataforma que permita estimar y predecir la cantidad de alumnos que tomarán un curso en la Escuela de Ingeniería en el siguiente periodo académico. Esto nace gracias al nuevo plan de estudios implementado por la Escuela de Ingeniería en el 2013, éste se encuentra dividido en 2 ciclos formativos: Licenciatura en Ciencias de la Ingeniería, correspondiente a los primeros 4 años, y articulación con un Título Profesional UC u otros grados académicos. La flexibilidad de esquema tiene como consecuencia que un curso puede estar en el camino académico de alumnos muy diversos, lo que se traduce en alta variabilidad en el número de alumnos que toman el curso.

Para gestionar la demanda de cursos dado los ciclos formativos en la Escuela de Ingeniería se diseñó una plataforma web utilizando el framework Ruby On Rails en conjunto con el entorno estadístico R. Este stack de tecnologías permite abordar de forma modular los desafíos a nivel de información para la creación de modelos predictivos y gestión de información a través de la plataforma web. Se enfrentó el proceso de levantamiento de requerimientos con la metodología de Relatos de Usuario y para el diseño de modelos predictivos se escogió la metodología Cross Industry Standard Process for Data Mining (CRISP-DM). A modo general, la plataforma web permite gestionar la información presente (cargar nuevos datos , actualizar y/o eliminación la información ya existente), gestionar los modelos predictivos para cada curso y visualizar lo resultados obtenidos.

Para concluir, se presentan los resultados de las predicciones en base a los datos proporcionados por Dirección de Pregrado, además de un listado de soluciones a los problemas que se presentan por la falta de datos dado que aún no existe una generación de alumnos que haya finalizado el primer ciclo.

1. CAPÍTULO 01

1.1. Motivación

La Escuela de Ingeniería de la Pontificia Universidad Católica desde año 2013 decidió implementar un nuevo plan de estudio dinámico y flexible basado en nuevas áreas de ingeniería e interdisciplina como foco de desarrollo (*Sitio Web Ingeniería - PUC2*, n.d.). Este nuevo currículum académico tiene dos ciclos de formación : Licenciatura en Ciencias de la Ingeniería y Articulación. La duración para el primer ciclo de formación es de 4 años y el segundo va desde 2 a 4 años dependiendo de la modalidad escogida por el estudiante.

La flexibilidad de esta modalidad de estudios se basa en el hecho de que alumno tiene múltiples opciones de formación en ambos ciclos. Dado que el primer ciclo se basa en una estructura T, la cual define un bloque de amplitud y otro de profundidad, el alumno debe seleccionar el conjunto de cursos para definir su especialización, denominado Major, y el conjunto de cursos denominados Minor con el objetivo de completar aún más su especialización o completar sus estudios en una área diferente a la enseñada en su Major. Al haber finalizado este primer ciclo de estudios, el alumno obtiene el grado de licenciado.

La formación del estudiante continúa con el segundo ciclo de formación, el cual posee los siguientes caminos:

(i) Articulación con un Título Profesional de Ingeniero UC.
(ii) Articulación con otros títulos profesionales UC. Hoy existen convenios para articular con los títulos de Médico Cirujano, Arquitecto y Diseñador UC.
(iii) Continuar con un grado académico superior de postgrado (magíster y doctorado). Este puede ser UC o no UC, y requiere de postulación independiente.
(iv) Emprendimiento o empleo temprano.

Uno de los principales desafíos que genera esta estructura dinámica y flexible es la gestión de recursos necesarios para la programación de los cursos dictados semestre a semestre, dentro de los recursos más valiosos podemos mencionar la gestión de la salas

para el uso de cátedras, laboratorios y ayudantías de los cursos así como también la planta de profesores necesarios para las diferentes secciones de cada curso.

Todo lo anterior genera la necesidad de conocer cuántos alumnos tomarán un determinado curso en el siguiente periodo académico. El presente trabajo busca dar la respuesta a esta problemática a través de la implementación de una plataforma web que permita la recolección de la nueva información generada por el nuevo currículum, la gestión de modelos predictivos y, finalmente, la entrega de resultados para poder tomar decisiones de cómo estructurar algunos de los recursos anteriormente mencionados, en particular, la planta de profesores.

1.2. Contexto

1.2.1. Estructura Interna de la Pontificia Universidad Católica de Chile

Para entender en el contexto el cual se desenvuelve el nuevo plan de estudio de Ingeniería UC, el cual tiene dentro de sus objetivos principales el cultivo de disciplinas interdisciplinarias, se hace necesario entender la estructura académica sobre la cual se rige la Pontificia Universidad Católica de Chile, en adelante UC.

Los siguientes puntos obtenidos del Reglamento sobre la Estructura Académica de la Universidad Católica de Chile (*Reglamento de Estructura Académica*, n.d.) permiten entender las entidades que conforman a la universidad y cuáles de ellas se encuentran autorizadas para proveer grados académicos y títulos profesionales chilenos:

(i) De acuerdo al punto I, artículo 2, se extrae lo siguiente:
   (a) La Pontificia Universidad Católica de Chile se estructura sobre la base de **unidades académicas** que son la organización fundamental para la cual la Universidad realiza sus actividades propias.

(b) Las unidades académicas a través de las cuales la Universidad realiza sus labores de docencia investigación y extensión son las **Facultades**, los **Institutos**, las **Escuelas** y los **Departamentos**.

(c) Las unidades académicas anteriormente mencionadas, a excepción de los Departamentos, podrán otorgar **grados académicos** y **títulos profesionales**, de conformidad con la reglamentación vigente.

(ii) De acuerdo al punto I, artículo 4, se enuncia:

(a) Las Facultades son las unidades académicas principales, organizadas en torno a una o más áreas del saber, con el propósito de coordinar las actividades de docencia, investigación y extensión, y asegurar una adecuada representación ante el Honorable Consejo Superior de la Universidad.

(b) Las Facultades pueden estar formadas por uno o más Institutos y Escuelas y dos o más Departamentos.

(c) Algunas unidades académicas podrán pertenecer a más de una Facultad, con el objeto de cumplir mejor sus fines y tener una estructura más adecuada y coherente con el trabajo interdisciplinario que realizan.

(d) Los Institutos y las Escuelas son unidades académicas de la Universidad dependientes de una Facultad.

(iii) De acuerdo al punto I, artículo 6, se entiende:

(a) Los Departamentos son unidades académicas de la Universidad, integradas por académicos que desarrollan actividades en torno a una misma disciplina o disciplinas afines del saber. Los Departamentos pueden depender de los Institutos o Escuelas o directamente de la Facultad a la que pertenecen.

Frente a los puntos anteriormente expuestos, podemos establecer el siguiente nivel jerárquico para las unidades académicas relacionadas con la Escuela de Ingeniería:

(i) La universidad está compuesta por Facultades.

(ii) Las Facultades están compuestas por Institutos y/o Escuelas.

(iii) Los Departamentos pueden depender de los Institutos o Escuelas o directamente de la Facultad a la que pertenecen.

El esquema anterior permite representar las unidades académicas existentes y la relación entre ellas. A continuación la tabla 1.1 se muestran las Facultades y las unidades académicas a su cargo.

Tabla 1.1. Facultades , Institutos, Escuelas y Departamentos en la UC

Facultad	Unidad Académica	Unidad Académica	Unidad Académica
Facultad de Agronomía e Ingeniería Forestal	-	-	-
Facultad de Arquitectura, Diseño y Estudios Urbanos	Escuela de Arquitectura	Escuela de Diseño	Instituto de Estudios Urbanos
Facultad de Artes	Escuela de Arte	Escuela de Teatro	Instituto de Música
Facultad de Ciencias Biológicas	-	-	-
Facultad de Ciencias Económicas y Administrativas	Escuela de Administración	Instituto de Economía	-
Facultad de Ciencias Sociales	Escuela de Psicología	Instituto de Sociología	Escuela de Trabajo Social
Facultad de Comunicaciones	-	-	-
Facultad de Letras	-	-	-

Tabla 1.1. Facultades , Institutos, Escuelas y Departamentos en la UC

Facultad	Unidad Académica	Unidad Académica	Unidad Académica
Facultad de Derecho	-	-	-
Facultad de Educación	-	-	-
Facultad de Filosofía	Instituto de Estética	Instituto de Filosofía	-
Facultad de Física	Instituto de Astrofísica	Instituto de Física	-
Facultad de Historia, Geografía y Ciencia Política	Instituto de Ciencias Política	Instituto de Geografía	Instituto de Historia
Facultad de Ingeniería	Escuela de Construcción Civil	Escuela de Ingeniería	-
Facultad de Matemática	-	-	-
Facultad de Medicina	Escuela de Enfermería	Escuela de Medicina	-
Facultad de Química	-	-	-
Facultad de Teología	-	-	-
Campus Villarica	-	-	-

1.2.2. Cursos

Un curso dentro de la universidad consiste en el instrumento académico a través del cual una unidad académica entrega un conjunto de habilidades y conocimientos relacionados a un tema o más temas en particular. Cada curso puede formar parte de diferentes programas de cursos cumpliendo un rol diferente en cada uno. Los roles que puede tener un curso son mínimo, optativo de profundización o electivo según el programa del alumno.

Por otro lado, cada curso exige un nivel de esfuerzo a realizar por el alumno, el cual se encuentra medido por sus créditos, los cuales representan la expresión cuantitativa del trabajo académico efectuado por el alumno, necesaria para alcanzar los objetivos y logros de aprendizaje del curso o actividad curricular (*Créditos UC*, n.d.). Este trabajo incluye clases teóricas o de cátedra, actividades prácticas , de laboratorio o taller, actividades clínicas o de terreno, estudio personal y evaluaciones. Un crédito UC equivale a una hora de trabajo por semana.(*Créditos UC*, n.d.)

1.2.3. Sistema de Créditos Académicos Transferibles

El plan de estudios de Ingeniería UC posee la particular de ser compatible con mallas curriculares a nivel internacional de manera de impulsar la movilidad de hacia postgrados locales e internacionales (*Sitio Web Ingeniería - PUC2*, n.d.).

Para alcanzar este objetivo, dentro de los programas de cursos ya mencionados anteriormente ,Major y Minor, el trabajo académico se expresa en una unidad académica denominada crédito SCT-Chile, la cual se rige bajo El Sistema de Créditos Transferibles, denominado SCT-Chile (*Sistema de Créditos Transferible Chile*, n.d.), el cual posee el mismo objetivo de créditos UC, poder cuantificar de forma racional el trabajo académico que un alumno debe dedicar a sus estudios en un año o semestre.

A continuación se muestra la tabla de equivalencia entre créditos SCT-Chile y créditos UC:

Tabla 1.2. Sistema de conversión de Créditos UC y Chileno

Sistema de créditos académicos transferibles SCT-Chile	Sistema de Créditos UC
30 créditos SCT-Chile al semestre	50 Créditos UC al semestre
Un año de estudios a tiempo completo equivale a 60 créditos SCT-Chile	Un año de estudios completo equivale a 100 créditos, lo cual es equivalente a 1800 horas de trabajo académico del estudiante

Fuente: http://admisionyregistros.uc.cl/alumnos/cursos/creditos-de-un-curso

De la tabla anterior 1.2, podemos extraer 1 crédito SCT-Chile equivale a 1.67 créditos UC.

1.2.4. Estructura del Plan de Estudios de Ingeniería UC

A continuación se entregan los detalles sobre la estructura que posee cada ciclo de formación, Licenciatura en Ciencias de la Ingeniería y Continuidad con un Título Profesional UC y/o articulación con otros grados académicos, así como empleo temprano y el emprendimiento, en el plan de estudios mencionado en los párrafos anteriores.

1.2.4.1. Licenciatura en Ciencias de la Ingeniería

Este ciclo formativo posee una duración de 4 años y posee los siguientes componentes (*Guía Alumno Ingeniería UC 2017*, n.d.):

(i) **Plan Común de Ciencias Básicas:** El objetivo de esta área es consolidar las bases en física, biología, química y matemática.

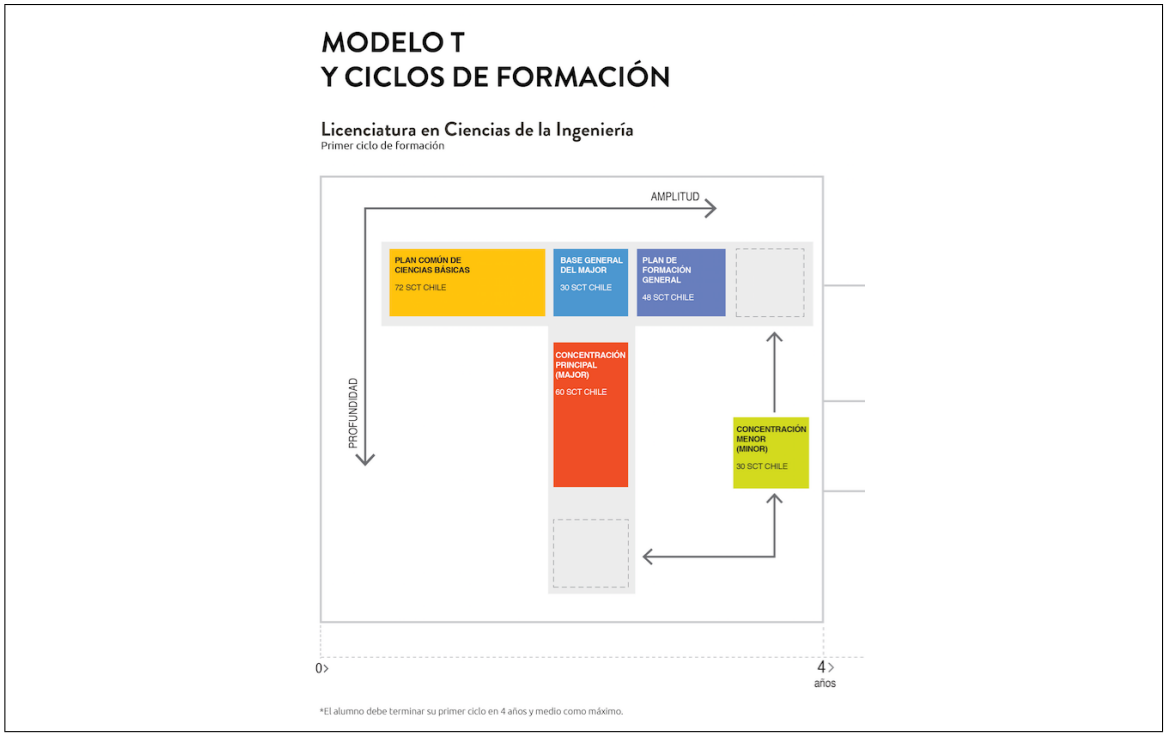
(ii) **Base General del Major:** Cimienta las bases para desarrollar las ciencias de la ingeniería.

(iii) **Plan de Formación General:** Cursos en disciplinas diferentes.

(iv) **Concentración Principal (Major):** Programa que entrega profundidad en un área disciplinaria o interdisciplinaria.

(v) **Concentración Menor (Minor):** Programa que permite ampliar o profundizar en un área disciplinaria o interdisciplinaria dependiendo del área escogida en el major.

La estructuración de los 5 elementos mencionados se basa en un modelo T definiendo una amplitud gracias al Plan Común de Ciencias Básicas, Base General del Major y Plan de Formación General, y otra de profundidad entregada por la Concentración Principal (Major). Este modelo T puede verse reflejado en la figura 1.1:



Fuente: (*Guía Alumno Ingeniería UC 2017*, n.d.)
Figura 1.1. Modelo T en el primer ciclo formativo

De la figura 1.1 cabe destacar que el Minor complementa el aspecto profundidad o amplitud dependiendo del área del Major escogido.

Dado el objetivo de compatibilizar con mallas curriculares internacionales, los ciclos formativos miden su exigencia académica en créditos SCT-Chile, los cuales puede ser transformados a créditos UC gracias a la equivalencia entregada por la universidad. La distribución de carga académica por componente tanto en créditos SCT-Chile como créditos UC se distribuye en la siguiente tabla:

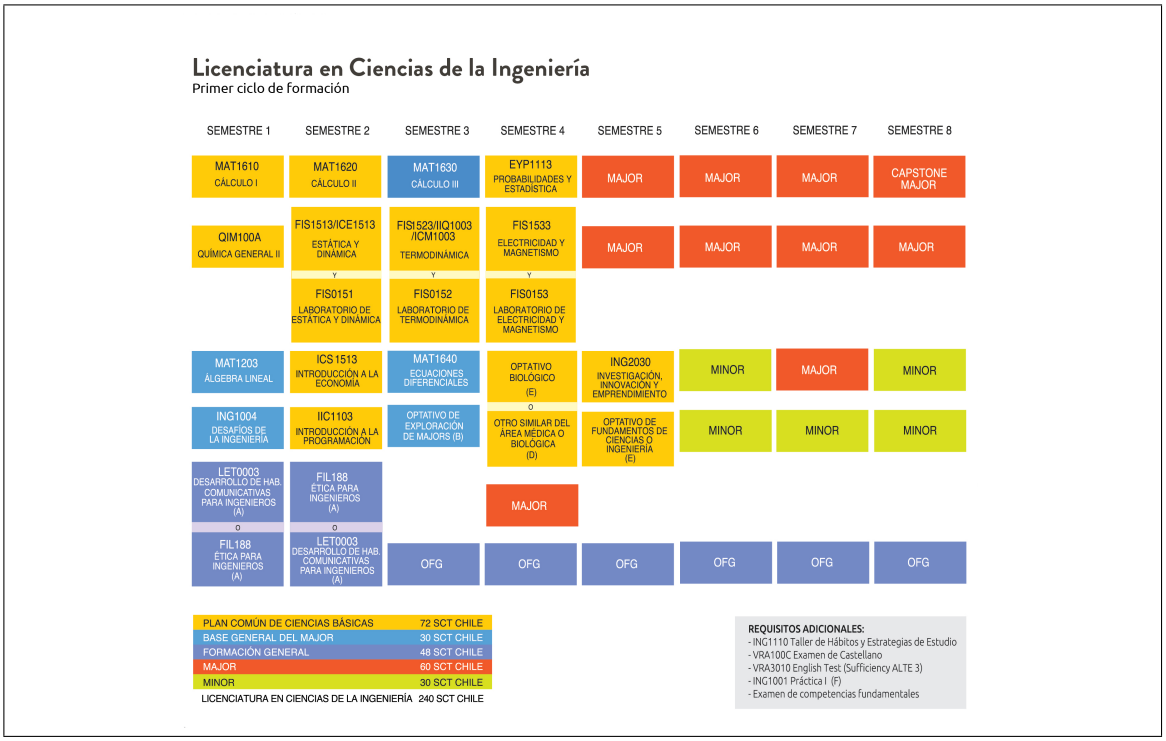
Tabla 1.3. Distribución de créditos SCT-Chile y UC en el primer ciclo formativo

Componente	Créditos SCT-Chile	Créditos-UC
Plan Común de Ciencias Básicas	72	120
Base General del Major	48	80
Plan de Formación General	48	80
Concentración Principal (Major)	60	180
Concentración Menor (Minor)	30	50

Esta estructura define la malla curricular del primer ciclo formativo, la cual puede ser visualizada en la ilustración 1.2.

La diversidad en la formación que puede escoger un alumno se encuentra en la conjunto Major-Minor que escoge, actualmente la Escuela de Ingeniería ofrece un total de 21 Majors, de los cuales 14 son disciplinarios y 7 interdisciplinarios, al mismo tiempo, entrega una oferta de 57 Minors entre los cuales, 32 son profundidad y 25 de amplitud. En el Anexo I se detalla los Majors y Minors disponibles.

Finalmente, con el objetivo de orientar a los alumnos hacia una elección de Major/Minor basada en sus intereses, se les solicita que hagan declaraciones de Major al término del segundo semestre y tercero para terminar con elección de Major y Minor al finalizar su cuarto semestre.



Fuente: (*Guía Alumno Ingeniería UC 2017*, n.d.)
Figura 1.2. Malla curricular primer ciclo formativo

1.2.4.2. Continuidad - Articulación Posterior a la Licenciatura

El segundo ciclo formativo se inicia una vez obtenida la Licenciatura en Ciencias de la Ingeniería, el cual entrega los siguientes caminos a seguir:

(i) Continuidad a un título profesional de Ingeniería Civil UC.

(ii) Continuidad a otros títulos profesionales UC. Hoy existen convenios para continuar estudios a los títulos de Médico Cirujano, Arquitecto y Diseñador UC.

(iii) Articulación al título profesional Ingeniería UC y al Magíster en Ciencias de la Ingeniería o el Doctorado en Ciencias de la Ingeniería UC.

(iv) Realizar un grado académico superior de postgrado (magister o doctorado) UC u otra.

(v) Salida al Mercado Laboral: Emprendimiento o Empleo Temprano.

1.3. Desafíos de información identificados

El plan de estudios descrito recientemente fue implementado el año 2013, esto trae consigo una serie de cambios tanto en la comunidad de estudiantes por los nuevos conceptos establecidos como a nivel administrativo para la planificación de cursos para los diferentes ciclos y profesores para los cursos impartidos en cada uno. Esta situación se vuelve particularmente crítica en particular por varios cambios realizados en la planificación inicial de cada Major y/o Minor, así como también en las Articulaciones establecidas entre la Escuela de Ingeniería y otras unidades académicas como Diseño y Medicina. Tanto los cambios como la propia estructura flexible levantan un desafío a nivel de información en cómo realizar un seguimiento adecuado del plan de estudio que realiza cada alumno, para poder disponer de la mejor manera los recursos disponibles por la Escuela.

Las necesidades de información que se detectan para realizar un seguimiento adecuado del plan de estudios de cada alumnos pueden ser descritas en las siguientes categorías:

(i) Registro de cursos tomados por los alumnos en cada periodo académico.
(ii) Registro de las declaraciones de Major realizadas por alumnos por periodo académico.
(iii) Gestión de demanda de cursos para cursos, Major y Minor por parte de los alumnos de las diferentes generaciones del nuevo plan.

2. CAPÍTULO 02 : MARCO TEÓRICO

2.1. Introducción

El presente trabajo se basa en conocimiento de Ingeniería de Software *(Software Engineering)* y el proceso denominado Descubrimiento del Conocimiento *(Knowledge Discovery)*. En el presente capítulo se exponen las definiciones de ambos campos, así como también los conceptos de cada área del conocimiento en Ciencias de la Computación.

2.2. Ingeniería de Software

De acuerdo a la *Association for Computer Machinery* (*Association for Computing Machinery*, n.d.), de ahora en adelante A.C.M., Ingeniería de Software o *Software Engineering*, en adelante S.E., se preocupa de la construcción y mantención de software que posee las siguientes características:

(i) Comportamiento confiable y eficiente.
(ii) Existe el modo de poder seguir desarrollando sobre ellos y al mismo tiempo mantenerlos.
(iii) Satisfacen todos los requerimientos definidos por las necesidades de los clientes.

Por su parte, *Institute of Electrical and Electronics Engineers*, de ahora en adelante I.E.E.E. (P. & R., 2004) define S.E. como la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de software, es decir, la aplicación de ingeniería al software.

2.2.1. Modelos y Ciclos de Vida del Desarrollo de Software

A lo largo de la historia de S.E. se ha creado una diversidad de modelos o ciclos de vida para la creación y mantención de software. Cada modelo define la forma o estrategia en que las distintas actividades del proceso deben ser llevadas a cabo para el desarrollo

y mantención del producto de software. A continuación se analizan los enfoques más descritos en la literatura actual.

2.2.1.1. Modelo Cascada

Este enfoque define una secuencia lineal de pasos para la creación de un software. Las etapas suelen incluir análisis de requisitos del sistema, análisis de requisitos de software, diseño preliminar, diseño, codificación, pruebas y mantenimiento. La siguiente ilustración muestra las etapas mencionadas anteriormente.

!!!! FIGURA !!!!

El modelo de cascada se caracteriza por sus fases secuenciales, las cuales deben terminar de modo completo antes de iniciar una nueva. Adicionalmente, posee un fuerte enfoque en el levantamiento de requerimientos antes de iniciar cualquier actividad de código o de implementación del software, por lo cual se caracteriza por una intensa comunicación con los clientes o stakeholders en las fases iniciales, para luego dar protagonismo a las fases de codificación, pruebas y mantenimiento.

Ventajas

(i) Simple, permite una clara visualización de avance del proyecto.
(ii) Se genera buena documentación que permite una clara planificación de los eventos dentro del proyecto.
(iii) Etapa de codificación sólo se enfrenta después de tener un diseño robusto.

Desventajas

(i) No es flexible frente a los cambios y deseos del cliente.
(ii) Los errores sólo pueden ser detectados en etapas tardías de implementación.
(iii) Cambios en los requerimientos agregan altísimos costos adicionales.

2.2.1.2. Modelo Incremental

Este modelo consiste en una evolución del anterior mejorando la flexibilidad y al mismo tiempo disminuyendo el tiempo de interacción con los stakeholders a lo largo de todo el proyecto. Con estos objetivos en mente, en vez de crear un proceso con etapas aisladas, se crea un proyecto que puede ser diseñado, desarrollado e implementado en etapas sucesivas, incorporando al mismo tiempo la retroalimentación del cliente en etapas intermedias (Massey & Satao, 2012). El siguiente esquema ejemplifica la filosofía de este nuevo enfoque:

!!!! FIGURA !!!!

Ventajas

(i) Incorpora flexibilidad al permitir incorporar necesidades que no fueron previstas o consideradas en la etapa inicial del proyecto.
(ii) Permite el desarrollo progresivo del proyecto hasta que está completo validando con los stakeholders durante todo el proyecto y no cuando éste haya finalizado.

Desventajas

(i) La flexibilidad permitida por el proyecto puede hacer el proyecto más costoso debido a los constantes cambios que pueden realizarse.
(ii) Al incorporar nuevos elementos, debido a las etapas incrementales, puede tener problemas de incompatibilidad con versiones anteriores del software.

2.2.1.3. Rapid Application Development (RAD)

Esta filosofía de desarrollo se basa en la idea de que los métodos desarrollados hasta los inicios de los 90 símplemente eran muy rígidos, en consecuencia no se podían aplicar de forma eficiente cuando las fechas de entrega cobran importancia. La ideología se basa en la entrega rápida de software mientras se mantiene una alta calidad en la implementación

desarrollada. Esta metodología fue desarrollada y formalizada por James Martin en el mismo horizonte de tiempo.

Este proceso de creación posee dos modalidades, la primera consiste en 4 fases secuenciales de tiempo : planificación de requerimientos, diseño desde el punto de vista del usuario, construcción y transición a la siguiente fase. En la segunda modalidad se fusionan las fases de planificación de requerimientos y diseño en una sola de análisis. Todas las fases de desarrollo en ambas modalidades se desarrolla en un espacio de tiempo determinado denominado timebox, donde se priorizan las actividades en función de las necesidades del negocio sin importar el desafío técnico al cual se ven enfrentadas (Gottesdiener, 1995). A continuación se presenta un posible esquema de trabajo bajo esta metodología:

!!!!! FIGURA !!!!!!

Ventajas

(i) Al enfocarse en prioridades de negocio sin importar dificultades técnicas permite entregar una implementación que satisface las necesidades más críticas de los *stakeholders*.

(ii) Al tener un tiempo específico de desarrollo se mantiene un paso a producción marcado por hitos.

(iii) Debido a que los espacios de tiempo de desarrollo entre bloques no son necesariamente del mismo horizonte temporal es posible adaptarlos a las nuevas necesidades que vayan surgiendo en el transcurso del proyecto.

(iv) Posee una comunicación constante para el desarrollo de cada fase.

Desventajas

(i) Sólo puede ser usada cuando las personas que lo desarrollan tienen un alto nivel técnico y una gran capacidad de trabajo en equipo, debido a que los tiempos de desarrollo necesitan ser lo más preciso posible por cada bloque.

(ii) Es posible que se creen expectativas falsas por parte de las personas que gestionan el proyecto al ser trabajado bajo espacios específicos de tiempo, lo que podría generar discusiones internas con el equipo de desarrollo.

2.2.1.4. Enfoque Ágil

Esta modalidad de desarrollo nace específicamente en el año 2001 con la publicación del Manifiesto Ágil, en donde se da a conocer un *framework* de cómo construir un software en un ambiente de constantes cambios y al mismo tiempo mantener un alto nivel de calidad de implementación y satisfacción al cliente. Existen 12 principios declarados en este manifiesto, los cuales pueden ser resumidos en las siguientes declaraciones (Isaias P., 2015):

(i) La satisfacción del cliente es la principal prioridad.
(ii) Cambios en los requerimientos son bienvenidos, no son más un obstáculo.
(iii) El *software* es entregado regularmente en constantes *release*.
(iv) Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
(v) Individuos motivados son una pieza vital para el éxito del proyecto.
(vi) Conversaciones cara a cara es esencial para una exitosa colaboración.
(vii) El *software* funcionando es la medida principal de progreso.
(viii) El desarrollo sostenible debe ser alentado.
(ix) Enfocado en bases técnicas y diseño de calidad.
(x) Simplicidad debe ser favorecida.
(xi) La mejor forma de gestionar un proyecto es con equipos auto organizados.
(xii) Debe haber constantes discusiones para el mejoramiento del equipo.

En términos generales es posible establecer 4 etapas para el desarrollo bajo este enfoque (Isaias P., 2015):

1. **Selección y aprobación del proyecto:** Los stakeholders (administradores, desarrolladores, clientes, entre otros) definen de modo conjunto el alcance, propósito y requerimientos del producto o implementación.
2. **Iniciación del proyecto:** El equipo de trabajo es construido en base a los requerimientos anteriormente establecidos. Adicionalmente, se establece la infraestructura de trabajo y se procede a la instalación de las herramientas tecnológicas a ocupar. En general, se establecen espacios de tiempo de trabajo para el equipo y reuniones de acuerdo a las necesidades del proyecto.
3. **Construcción basada en iteraciones:** Las iteraciones consisten en dos etapas, planificación y construcción. La idea es que al final de cada iteración se obtenga una pieza de software terminada.
4. **Entrega de producto maduro *(product release)*:** Esta fase final está caracterizada nuevamente por dos etapas, pruebas y mejoramiento así como también de la documentación necesaria para el entendimiento del proyecto.

El siguiente esquema muestra el proceso genérico bajo cualquier proceso con enfoque ágil:

!!!! FIGURA !!!!!

2.2.2. Metodologías de Desarrollo de Software

Un modelo o ciclo de vida puede ser considerado como una estructura de referencia, una estrategia general para enfrentar el desarrollo, pero no prescribe el modo concreto de llevarlo a cabo. La forma específica de cómo llevar a cabo el proyecto la define una metodología orientada por un determinado modelo o enfoque.

Dentro de los modelos mencionados anteriormente él más relevante hoy en día es enfoque ágil debido a los cambios que ha sufrido la industria tanto a nivel tecnológico como social. Desde su nacimiento en el año 2001 por la declaración del Manifiesto Ágil han surgidos diversas metodologías, las cuales enfrentan diferentes desafíos impuestos

por los supuestos de esta modalidad de desarrollo, tales como incorporación de cambios en cualquier momento del proyecto. A continuación se describen dos de las metodologías más populares que permiten implementar el enfoque ágil, éstas son Scrum y Extreme programming (XP). La figura muestra la dominancia de Scrum entre las metodologías ágiles:

!!!!! FIGURA !!!!!

Como se aprecia sólo Scrum cubre más del 50% de la industria. Si bien XP representa sólo un 1%, es la base de Scrum/XP Hybrid que es usado por 10% de la industria y es muy usada en grupos pequeños de desarrollo. Por lo tanto, en el desarrollo de este trabajo se consideraron tanto Scrum como XP.

2.2.2.1. Scrum

Esta metodología se basa principalmente en cómo un equipo debe funcionar y las relaciones que los miembros de éste establecen. Para lograr este objetivo se definen roles, eventos, artefactos y reglas. Cada componente dentro de este marco de referencia sirve para un propósito específico que son esenciales para éxito de su aplicación (Schwaber K., 2013). A continuación se ilustra la forma de trabajo en esta metodología en dos aspectos, primero se identificación los elementos presente en la metodología y a continuación se muestra cómo estos interactúan en el transcurso de un proyecto:

!!!!! FIGURA !!!!!

!!!!! FIGURA !!!!!

Los roles identificados en este metodología son:

1. *Stakeholder*: Corresponde a todas aquellas personas que se involucran en el proyecto, desde los usuarios finales hasta quienes financian el desarrollo del proyecto. Se identifica a todos aquellos que poseen algún interés una vez que el proyecto esté finalizado.

2. *Product Owner*: Es el encargo de entender el contexto en el cual se desenvuelve el proyecto y en base a eso llevar a cabo el levantamiento de requerimientos del proyecto.

3. *Scrum Master*: Es el responsable de verificar que la metodología Scrum sea aplicada de forma correcta, así como también el encargado de proporcionar a cada miembro del equipo los elementos que necesite para llevar a cabo su tarea. Este rol debe estar al servicio tanto del *Product Owner* como para los miembros del equipo de desarrollo.

4. *Development Team*: Son los responsables de la creación del proyecto en base a los requerimientos entregados por el *Product Owner*.

Por su parte, la definición de los artefactos en esta metodología son:

1. *Product Backlog*: Es una lista ordenada de todas las necesidades del producto/proyecto y es la única fuente de requerimientos existente. El responsable de su escritura es el Product Owner, quien debe procurar por su contenido, disponibilidad y orden. Se debe indicar que nunca está completo y permanece en frecuente cambio frente a los diferentes requerimientos que el proyecto pueda ir necesitando. A modo general este artefacto posee las funcionalidades, requerimientos, mejoras y reparaciones para las próximas versiones del producto. Los elementos presentes poseen nombre o título, atributos que permiten describirlo, orden, estimación y valor para el proyecto (Schwaber K., 2013).

2. *Sprint Backlog*: Es el conjunto de ítems seleccionados del Product Backlog que conforman el siguiente entregable del producto/proyecto que se está desarrollando. Se puede interpretar como la proyección de lo que será el software en su siguiente fase, donde todos los elementos presentes en este conjunto se considerarán terminados.

3. *Increment*: Es la suma de todos los elementos desarrollados en un Sprint Backlog. Este debe considerarse terminado y listo para ser usado en la próxima versión del software.

Finalmente, se definen eventos dentro de Scrum con el objetivo de mantener una comunicación regular y eliminar la necesidad de encuentros. Cada evento posee una duración limitada y que puede varias según el proyecto así como también del equipo quien implemente la metodología, sin embargo, existen un periodo de tiempo recomendado en cada caso.

Los eventos presentes en Scrum son (Schwaber K., 2013):

1. *Sprint*: Se considera como el corazón de Scrum debido a que es el periodo de tiempo base en el cual se entrega un increment (incremento) del software. La duración recomendada para un sprint es un mes o semanas y es ideal que mantenga la misma duración durante todo el proyecto. Dentro de este periodo de tiempo ocurre el sprint planning, daily scrum, sprint review y, finalmente, sprint retrospective.

2. *Sprint Planning*: Periodo de tiempo en el cual se planifica el trabajo y orden del mismo a realizarse en durante el sprint. El tiempo máximo para un sprint planning es de 8 horas cuando el tiempo del sprint es de un mes. El scrum master es el responsable de la gestión y la ejecución de esta fase. Por otro lado, se entiende que este periodo tiene por objetivo responder ¿qué elementos formarán parte del próximo increment ? y ,al mismo tiempo, ¿qué trabajo se realizará para cumplir el requerimiento descrito en el Product Backlog?

3. *Dialy Scrum*: Consiste en una reunión de 15 minutos de duración donde el equipo de desarrollo sincroniza y crea el plan de desarrollo para las próximas 24 horas. Este trabajo se hace en base a la última

reunión realizada. Adicionalmente, los miembros del equipo explican qué hicieron en el último daily scrum, que harán en el actual y de forma conjunta identifican e informan los problemas que han ocurrido y los que podrían ocurrir para el desarrollo del siguiente daily scrum.

4. ***Sprint Review*****:** Es una reunión informal con máxima duración de 4 horas donde los miembros del equipo de desarrollo identifican los logros obtenidos al final del sprint. Es una instancia en donde es posible modificar el Product Backlog. En esta reunión deben participan stakeholders invitados por el Product Owner y todos los miembros de la metodología. Los principales objetivos de este intervalo de tiempo son:

(i) Qué se hizo y qué no hizo.

(ii) Se explican en detalle los problemas ocurridos y cómo fueron resueltos.

(iii) Se discute de modo general qué elementos son importantes en base al avance logrado para el siguiente sprint planning.

(iv) Se discute el estado del Product Backlog y se modifica en base al incremento logrado, a los cambios provenientes de los stakeholders así como también de los eventos externos del proyecto.

5. ***Sprint Retrospective*****:**

Es un encuentro donde el equipo de desarrollo analiza su modo de trabajo y crea un plan de mejoras para ser atacadas en el próximo sprint. Se inspeccionan todos los ámbitos dentro de un equipo los miembros involucrados en el proyecto, relaciones entre ellos, procesos y herramientas utilizadas. Esta reunión se recomienda una duración máxima de 3 horas y se realiza al finalizar el sprint.

2.2.2.2. Extreme Programming

Esta metodología basa su funcionamiento en 2 principios principales: simplicidad y eficiencia. Con el objetivo de lograr estos principios se definen los siguientes 4 valores o elementos, los cuales poseen sus propias características:

(i) **Comunicación (Communication)**

Enfocada en lograr la cooperación y buena relación entre los miembros del equipo, para lograr esto se propicia la participación constante del cliente, programación a pares y poseer dentro del equipo de desarrollo una estandarización de código.

(ii) **Diseñar de modo simple (Simplicity)**

Orienta a que los elementos que se crean sean simples y basado en el pensamiento asociativo usando el mundo real como referencia, es decir, el uso de metáforas. Esto incluye las mejoras que se realizan a través de *refactoring* (Anexo A - 1).

(iii) **Retroalimentación constante (Feedback)**

Consiste en definir los modos para asegurarse que el software es aceptado tanto por el equipo de desarrollo como los clientes. Para ello necesita de testing continuo, integración continua (recomendado al menos una vez al día) y ,finalmente, basar las entregas en pequeñas funcionalidades.

(iv) **Aceptación de nuevos cambios (Courage)**

Consta de todas las acciones usadas para la organización del proyecto y del equipo, para ello motiva realizar el levantamiento de requerimientos en base a relatos de usuario para el proyecto y a nivel de equipo estimula el concepto de propiedad de código colectiva.

Los tiempos de desarrollo en XP se basan en iteraciones de 3 semanas en las cuales se recomienda seguir las siguientes 12 prácticas de software:

(i) Planificación del proceso : En esta instancia es donde se recomienda usar relatos de usuarios para la descripción de las funcionalidades del software.

(ii) Entregables pequeños.

(iii) Entregables pequeños.

(iv) *Test Driven Develpment* (Anexo A - 2).

(v) *Refactoring* (Anexo A - 1).

(vi) Mantener el diseño simple.

(vii) Programación a Parea (Anexo A - 3).

(viii) Propiedad colectiva del código.

(ix) Uso de estándares de programación.

(x) Integración continua (Anexo A - 4).

(xi) Mantener al cliente cercano.

(xii) Mantener un buen clima laboral.

(xiii) Utilización de metáforas para el diseño : Consiste en crear un lenguaje común para el equipo de desarrollo y clientes con el objetivo de que todos puedan comprender los conceptos y definiciones usadas en el proyecto del mismo modo.

La metodología es poco prescriptiva (en comparación a Scrum) pero existen prácticas sugeridas para organizar el flujo de trabajo. Por ejemplo, el sitio ExtremProgramming.Org sugiere el siguiente esquema:

!!!! FIGURA !!!!

2.2.3. Levantamiento de Requerimientos

De acuerdo al Glosario Estándar de Ingeniería de Software de la IEEE (IEEE, 1990), el concepto de requisito puede entenderse de acuerdo a alguna de las siguientes definiciones:

(i) Condición o capacidad necesaria por un usuario para solucionar un problema o alcanzar un objetivo.

(ii) Condición o capacidad que debe ser encontrada o poseer un sistema o algún componente de éste que satisfaga un contrato, un estándar, especificación u otra formalidad impuesta por documentación.

Al mismo tiempo, de acuerdo a la guía PMBOK (Institute, 2013), identifica 3 categorías:

(i) **Requerimientos del Negocio**
Describen las necesidades de alto nivel de la organización como un todo, tales como, oportunidades o problemas del negocio, y los motivos por los cuales un proyecto ha sido iniciado.

(ii) **Requerimientos de los *Stakeholders***
Describen las necesidades de uno o varios *stakeholders*.

(iii) **Requerimientos de la Solución**
Describen las funcionalidades, funciones o características de un producto, servicio o resultado que resuelve los requerimientos del negocio y de los stakeholder. Dentro de los requerimientos de solución es posible encontrar dos tipos:

(a) **Requerimientos funcionales:** Describen el comportamiento del producto o desarrollo. Por ejemplo: procesos, información, base de datos e interacciones del usuario y producto.

(b) **Requerimientos no funcionales:** Entrega los criterios medibles a través de los cuales un sistema puede comprobar su funcionamiento. Las principales áreas en las cuales se definen estos criterios son rendimiento, nivel de seguridad, escalabilidad, entre otros.

Por lo descrito anteriormente, las metodologías permiten la captura de requisitos de diferentes maneras, las más usuales para los requerimientos del negocio y *stakeholders* corresponden a *focus groups* (Anexo A - 5), reuniones y encuestas. De modo transversal a las metodologías, este procedimiento de entendimiento donde se desarrolla el producto o *software* finaliza con un objetivo principal y una serie de objetivos secundarios que deben

cumplirse al término de éste. Esta dinámica en las metodologías mencionadas anteriormente se realiza en una etapa específica o designa un responsable para ello, por ejemplo, en Scrum el responsable de este procedimiento durante todo el proyecto es el *Product Owner*. Adicional a esto, al momento de especificar los requerimientos de solución, debido a que los requisitos funcionales describen comportamiento, el cual dentro de lo posible debe ser interpretado del mismo por todo los miembros del equipo, presentan la necesidad de realizarse bajo una estructura o esquema de descripción específico. Esta situación no sucede con los requerimientos no funcionales, debido a que ellos presentan métricas y sus criterios para ser evaluadas , por ejemplo, el tiempo de carga al sitio inicial del *software* no debe ser mayor a 1 segundo.

Dentro de las técnicas de descripción o **levantamiento de requisitos funcionales**, se encuentran Casos de Usos *(Use Cases)* y Relatos de Usuarios *(User Stories)*.

2.2.3.1. Casos de Uso

De Acuerdo a Cockburn (Cockburn, 2000), un caso de uso se define como la descripción del comportamiento del sistema bajo varias condiciones que responden a la necesidad de uno o varios *stakeholders*. Adicionalmente, se espera que un caso de uso sea fácil de leer, para una completa identificación de las necesidades se propone las siguientes partes para una descripción completa:

(i) **Nombre *(Name)***

Correspondiente al nombre de la acción que se desea realizar.

(ii) **Objetivo en Contexto *(Goal in Context)***

Menciona las motivaciones del actor principal para la realización de sus acciones, así como también la descripción del entorno en el cual las realiza.

(iii) **Alcance del diseño *(Design Scope)***

Corresponde a la definición de que se desarrolla y no desarrolla dentro del proyecto. Con el objetivo de describir el alcance dentro de un contexto Cockburn define los siguientes niveles o tipos de alcance:

(a) **Corporativo *(Corporative)*:** Indica que la discusión para definir el alcance envuelve al comportamiento de toda la organización. Por ejemplo, cuando se debe entregar un desarrollo a un agente externo al contexto del proyecto.

(b) **Sistema *(System)*:** Corresponde sólo a una parte del producto o software que se está construyendo.

(c) **Sub sistema *(Sub system)*:** Si bien Cockburn indica que a un alcance de sistema debería bastar para la descripción de un software. En caso de que se requiera dividir una pieza de software en varias partes recomienda ocupar la categoría de sub sistema para ella.

(iv) **Niveles de los objetivos**

Corresponde al nivel de especificidad que se declara en un objetivo.A continuación se enuncian los niveles existentes:

(a) ***Very High Summary*** Indica la razón principal de las acciones del usuario principal.

(b) ***Summary*** Describe un global más global que puede abarcar diferentes visiones provenientes de diferentes actores principales.

(c) ***User-goal level*** Corresponde a una descripción amplia de lo que se desea lograr. Esta descripción o deseo se hace a nivel del actor principal del caso de uso, por lo tanto, indica un nivel de agrupación de objetivos secundarios identificados como *subfunction*.

(d) ***Subfunction*** Corresponde al conjunto de objetivos obligatorios que deben cumplirse poder alcanzar otros objetivos de nivel superior. A este nivel un objetivo secundario ya es posible identificarlo como una acción clara del usuario.

(e) ***Too Low*** Corresponde a un nivel de objetivos más abajo que subfunction y agrupan todas las tareas más básicas que deben realizarse lograr el objetivo planteado.

El siguiente esquema permite visualizar de forma gráfica un ejemplo de los niveles mencionados:

!!!! FIGURA !!!!!

(f) **Actor Principal *(Principal Actor)*:** Indica al usuario que realiza o ejecuta el conjunto de acciones indicadas por el caso de usuario. Si bien Cockburn menciona en esta versión de caso de uso sólo un actor dentro del caso de uso, es posible incluir otros como actores secundarios.

(g) ***Stakeholders* e intereses *(Stakeholders and Interests)*:** Se atribuye al conjunto de stakeholder y los objetivos que cada uno busca.

(h) **Precondiciones *(Preconditions)*:** Corresponden al conjunto de elementos que se deben cumplir antes de iniciar el flujo de etapas del caso de uso.

(i) **Estado de éxito *(Success End Condition)*:** Muestra el estado en el cual el sistema se encuentra de la una ejecución exitosa del caso de uso.

(j) **Estado de error *(Failed End Protection)*:** Describe el estado del sistema en caso de no alcanzar el objetivo.

(k) ***Trigger*:** Corresponde a la acción que desencadena el caso de uso.

(l) **Descripción *(Description)*:** Conjunto de pasos a seguir para alcanzar el objetivo del caso de uso.

(m) **Extensiones *(Extensions)*:** Corresponde a un flujo adicional que puede ser el caso de uso al momento de darse un condición especial sobre el sistema o los actores involucrados.

(n) **Variaciones *(Variants)*:** Indica la ejecución de una misma acción de un modo distinto dentro del sistema , por ejemplo, al momento de pagar puede realizarse a través de dinero en efectivo, transferencia bancaria, entre otros.

La lista anteriormente descrita puede verse resumida en el siguiente recuadro conocido por Cockburn como la versión completamente vestida (fully dressed):

!!!! FIGURA !!!!!

Adicionalmente a lo descrito anteriormente, con el objetivo de tener una mejor visualización de la interacción entre casos de usos y actores principales como secundarios el

Unified Modeling Language (U.M.L.) (Rumbaugh J., 2004) entrega los estándares y elementos para realizar la diagramación de un caso de uso. Este tipo de diagrama se denomina use case view o vista de un caso de uso. Los elementos presentes en esta visualización son los siguientes:

!!!! FIGURA !!!!!

De este modo, para lograr la descripción de un caso de uso, se necesita tanto su diagrama descrito en UML como su detalle, el cual puede estar basado en la estructura propuesta por Cockburn o una estructura similar, pero más simple.

2.2.3.2. Relatos de Usuarios

Según Mike Cohn (M., 2004), un relato de usuario describe una funcionalidad que es valorada por un usuario o otro sistema que se alimenta de ella. Las partes de un relato consisten en la identificador,descripción de la funcionalidad, los criterios de aceptación, prioridad para el negocio y el puntaje de dificultad identificado por el equipo desarrollador. Adicional a lo anterior, cada relato de usuario debe cumplir las siguientes propiedades:

(i) **Independiente:** Un relato de usuario es independiente de los demás.

(ii) **Negociable:** No son contratos , por lo que puede ser discutida y descrita por todos los *stakeholders* y los miembros del equipo desarrollador.

(iii) **Valorada por los compradores y por los usuarios:** Indica que las características del software descrito debe agregar valor tanto para las personas que lo ocupan como aquellas que "pagan" por él.

(iv) **Estimable:** Es importante poder asignarle un tamaño al relato de usuario para identificar la cantidad de tiempo que tomará. Se indica que un relato que no es posible estimarlo es demasiado grande y es recomendable subdividirlo en varios.

(v) **Pequeña:** Indica que la descripción de un relato de un usuario no puede ser tan general porque en caso de ser así no podrá ser estimada de forma adecuada. En caso de suceder esto, se recomienda agruparlas bajo un epic (Anexo A - 6) y que

la funcionalidad sea dividida en relatos de usuarios más pequeños y con mayor cantidad de detalle.

(vi) **Comprobable:** Indica que debe existir una forma concreta y no general de que el relato de usuario se desarrolló de forma exitosa.

Con el objetivo de que el relato de usuario pueda cumplir con las propiedades descritas anteriormente, Cohen recomienda utilizar el formato de tarjeta para su descripción. La siguiente imagen ilustra este concepto:

!!!!!! FIGURA !!!!!!

(i) **Identificador:** Compuesto por un número o nomenclatura con el formato US-XX, donde XX corresponde al número del relato.

(ii) **Título:** Describe la funcionalidad que se debe desarrollador. El título debe posee el siguiente formato

As a **<type of user>**, I want **<some goal>** so that **<some reason>**.

Cohen (M., 2004) indica que la parte <some reason> en muchos casos es opcional.

(iii) **Puntaje:** Correspondiente al grado de dificultad del relato. Se espera que un relato de 2X tome el doble de tiempo que uno de X puntos.

(iv) **Prioridad:** Indica el orden en el cual deben ser desarrolladas los relatos de usuarios.

(v) **Criterios de Aceptación:** Corresponde a las condiciones que debe cumplir el relato de usuario. No se especifica un formato para escribir un criterio de aceptación, pero se recomienda que indique de forma precisa lo que se espera del software, por ejemplo, el sistema debe solicitar el nombre, apellido y correo electrónico para poder identificar a un usuario.

Otro punto importante que destaca Cohn es el uso de *epics* (Anexo A - 6), él indica que no existe un criterio universal para clasificar una historia de usuario como grande, pero sí deben buscar cumplir del mejor modo las propiedades anteriormente descritas. Frente a la

necesidad de agrupar, dos o más relatos de usuarios bajo un mismo concepto es cuando se hace el uso de un epic, el cual posee simplemente un nombre relativo a la funcionalidad que describe,una descripción para dar más contexto a quien lee la documentación del proyecto y un código identificador para realizar mejor el agrupamiento de los relatos de usuarios descritos anteriormente (M., 2004).

2.2.4. Metodología y técnicas usadas en este trabajo

Dentro de las metodologías y técnicas para levantamiento de requisitos descritas anteriormente para este proyecto se escogió la metodología XP y levantamiento de requisitos funcionales vía relatos de usuario. Se consideró que para la implementación de Scrum se hacía necesario incorporar un Product Owner, y un Scrum Master además de los miembros de equipo de desarrollo. XP es menos prescriptivo y permite organizar el trabajo en función de la experiencia del equipo de desarrollo, siempre y cuando, se respeten los principios que ésta estipula. Otro factor que se tuvo en cuenta es que dada la considerable complejidad del problema, lo que buscábamos es más bien producir un primer prototipo o lo que se conoce como un *Minimal Viable Product*, en adelante MVP (Anexo A - 7). XP gracias a su principio de simplicidad (Simplicity) permite abordar de mejor manera este escenario.

Por otro parte, a nivel de levantamiento de requisitos si bien tanto Casos de Usos como Relatos pueden ser aplicados con metodologías ágiles, se escogió relatos de usuario debido a su naturaleza menos estructurada que privilegia la comprensión de los requisitos en base a conversaciones con los stakeholder Adicionalmente a lo anterior, las propiedades de los relatos de usuarios (independiente, negociable, valorado por los usuarios o compradores, estimable, pequeña y comprobable) permite un mejor registro de los esfuerzos invertidos en el desarrollo del prototipo.

Finalmente, el hecho de que el formato de los relatos de usuarios marque explícitamente el objetivo que se buscar resolver permite una mejor visualización general de las

soluciones implementadas y al mismo tiempo permite llevar un control efectivo del avance del proyecto de modo más sencillo en base a cada uno de los artefactos requeridos por éste.

2.3. Descubrimiento de Conocimiento (Knowledge Discovery)

Fayyad (Mouhib Alnoukari, 2012) en 1996 define término Knowledge Discovery como el conjunto de conocimientos relativos al proceso de extracción, almacenamiento y accesibilidad de datos; el uso de algoritmos ,en forma eficiente, para el análisis de los mismos; y, finalmente, la interpretación y visualización de resultados en base a estos, este resultado es conocido como *knowledge*(conocimiento).

2.3.1. Categorías, metodologías y modelos

Desde la creación de este término hasta hoy en día han surgido varias metodologías para llevar a cabo el proceso anteriormente descrito, el cual se denomina *Knowledge Discovery Process (KDP)*. Mouhib y Asim (Mouhib Alnoukari, 2012) logran identificar un total de 4:

2.3.1.1. Metodologías y modelos

2.3.2. Metodologías y modelos usadas en este trabajo

3. CAPÍTULO 03

3.1. Figures and Tables

In Figure 3.1 we can see a selfie taken by a macaque.

Figura 3.1. This picture is not copyrighted

Blandit incorrupte quaerendum in quo, nibh impedit id vis, vel no nullam semper audiam. Ei populo graeci consulatu mei, has ea stet modus phaedrum. Inani oblique ne has, duo et veritus detraxit. Tota ludus oratio ea mel, offendit persequeris ei vim. Eos dicat oratio partem ut, id cum ignota senserit intellegat. Sit inani ubique graecis ad, quando graecis liberavisse et cum, dicit option eruditi at duo. Homero salutatus suscipiantur eum id, tamquam voluptaria expetendis ad sed, nobis feugiat similique usu ex.

And now, Table 3.1 shows some parameter for the Aliev-Panfilov model.

Tabla 3.1. Parameter values considered for the Aliev-Panfilov model of ionic current.

α	c_1	c_2	μ_1	μ_2	b	γ
0.05	52	8	0.1	0.3	0.25	0.002

There are 12 figures in order to show that the List of Figures works fine.

3.2. Equations

Finally, an equation

$$x^2 + 1 = 0 \tag{3.1}$$

We can notice that $x = \pm i$ are the solutions of Equation (3.1).

4. CONCLUSIONS

Nothing to say. Be happy.

REFERENCES

Agile User Stories,Epics and Themes. (n.d.). Retrieved from https://www.scrumalliance.org/community/spotlight/mike-cohn/march-2014/agile-user-stories-epics-and-themes

Association for Computing Machinery. (n.d.). Retrieved from http://computingcareers.acm.org/?page_id=12

Cockburn, A. (2000). *Writing Effective Use Cases*. Addison-Wesley Professional. Online.

Créditos UC. (n.d.). Retrieved from http://admisionyregistros.uc.cl/alumnos/cursos/creditos-de-un-curso

Definición de Focus Group. (n.d.). Retrieved from https://www.definicionabc.com/comunicacion/focus-group.php

Gottesdiener. (1995). *BUSCAR NOMBRE - Arreglar*. WHO KNOWS. Hardcover.

Guía Alumno Ingeniería UC 2017. (n.d.). Retrieved from https://issuu.com/ingenieriauc/docs/guia_del_alumno_2017

IEEE. (1990). *High Level Models and Methodologies for Information Systems*. Author. Online.

Institute, T. P. M. (2013). *A Guide to the Project Management of Body Knowledge (PMBOK GUIDE)*. IEEE. Online.

Isaias P., I. T. (2015). *High Level Models and Methodologies for Information Systems*. Springer. Hardcover.

M., C. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional. Online.

Massey, & Satao. (2012). *BUSCAR NOMBRE - Arreglar*. IEEE Computer Society, Los Alamitos, CA,USA. Hardcover.

Mouhib Alnoukari, A. E. S. (2012). *Chapter 4 Knowledge Discovert Process Models: From Traditional to Agile Modeling*. Business Science Reference (an imprint of IGI Global). Online.

P., B., & R., D. (2004). *Guide to the Software Engineering, Body of Knowledge - Arreglar*. IEEE Computer Society, Los Alamitos, CA,USA. Hardcover.

Reglamento de Estructura Académica. (n.d.). Retrieved from http://secretariageneral.uc.cl/images/Reglamento_sobre_Estructura_Acad%C3%A9mica_v__23_09_16.pdf

Rumbaugh J., B. G., Jacobson I. (2004). *The Unified Modeling Language Reference Manual (2nd Edition)*. Addison-Wesley Professional. Online.

Schwaber K., S. J. (2013). *The Scrum Guide*. Scrum.Org and ScrumInc. Hardcover.

Sistema de Créditos Transferible Chile. (n.d.). Retrieved from http://sct-chile.consejoderectores.cl/que_es_sct_chile.php

Sitio Web Ingeniería - PUC2. (n.d.). Retrieved from http://www.ing.uc.cl/alumnos/plan-de-estudios/descripcion/

¿Qué es la integración Continua. (n.d.). Retrieved from https://aws.amazon.com/es/devops/continuous-integration/

ANEXO

A. GLOSARIO

(i) ***Refactoring***

Es el proceso de reconstrucción de código existente sin cambiar su comportamiento. Mejora los aspectos non funcionales del *software*, por ejemplo, rendimiento.

(ii) ***Test Driven Development***

Proceso de desarrollo de software el cual se basa en la repetición de un ciclo de tres pasos. Primero, el desarrollador escribe un test automático que define el comportamiento deseado del software; segundo, se escribe la mínima cantida de código que hace el test pasar y ,finalmente, se hace un proceso de *refactor* del código.

(iii) **Programación a pares**

Proceso de programación donde dos programadores participan en un esfuerzo combinado de desarrollo en un sitio de trabajo. Cada miembro realiza una acción que el otro no está haciendo actualmente: Mientras que uno codifica las pruebas de unidades el otro piensa en la clase que satisfará la prueba, por ejemplo.

(iv) **Integración continua**

Práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización (p. ej., CI o servicio de versiones) y un componente cultural (p. ej., aprender a integrar con frecuencia). Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software (*¿Qué es la integración Continua*, n.d.).

(v) **Focus Group**

Tipo de técnica de estudio empleada en las ciencias sociales y en trabajos comerciales que permite conocer y estudiar las opiniones y actitudes de un público determinado (*Definición de Focus Group*, n.d.).

(vi) **Epic**

Agrupación de relatos de usuarios. También son considerados relatos de usuarios más grandes (*Agile User Stories,Epics and Themes*, n.d.).

(vii) **Minimal Viable Product**

Producto con suficientes características para satisfacer a los clientes iniciales, y proporcionar retroalimentación para el desarrollo futuro.

B. AN INTERESTING SHORT STORY

Let us enjoy reading this story of Hunting With The Lion.

It was a dry summer. The animals in the forest were beginning to find it difficult to get food.

A bear, a wolf and a jackal thought it would be better to join hands with a lion and do the hunting. They approached lion and he too agreed. The four of them went off hunting.

The hunting party came across a buffalo. The fox and wolf chased the buffalo. The bear intercepted the buffalo. The lion killed him.

The fox made shares out of the buffalo. When they were about to take their shares the lion roared and said, "Well friends, the first share is mine for my leadership. The second share is mine for, it is I who killed. The third share is also mine for I need it for my cubs. Anyone who needs a share can take the fourth. But before that you will have to win me."

All the three left the place without a single word.

MORAL : **If you are might, you are right**.