


**Universidad Nacional Mayor de San Marcos**

**Universidad del Perú. Decana de América**

Dirección General de Estudios de Posgrado

Facultad de Ciencias Matemáticas

Unidad de Posgrado


# Dinámica de Proyectos de Desarrollo de Software Multicapa a partir de la Evolución Cuantitativa del Código Fuente

**TESIS**

Para optar el Grado Académico de Magíster en Matemática Aplicada con mención en Matemática Computacional


**AUTOR**

Diego BRAVO ESTRADA

**ASESOR**

Roxana LÓPEZ CRUZ



Lima, Perú

2023




Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

https://creativecommons.org/licenses/by-nc-sa/4.0/

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

**Referencia bibliográfica**

Bravo, D. (2023). *Dinámica de Proyectos de Desarrollo de Software Multicapa a partir de la Evolución Cuantitativa del Código Fuente*. [Tesis de maestría, Universidad Nacional Mayor de San Marcos, Unidad de Posgrado]. Repositorio institucional Cybertesis UNMSM.

## Metadatos complementarios

| | |
|---|---|
| **Datos de autor** | |
| Nombres y apellidos | Diego Bravo Estrada |
| Tipo de documento de identidad | DNI |
| Número de documento de identidad | 25772463 |
| URL de ORCID | https://orcid.org/0000-0002-7439-0772 |
| **Datos de asesor** | |
| Nombres y apellidos | Roxana López Cruz |
| Tipo de documento de identidad | DNI |
| Número de documento de identidad | 10798114 |
| URL de ORCID | https://orcid.org/0000-0002-7703-5784 |
| **Datos del jurado** | |
| **Presidente del jurado** | |
| Nombres y apellidos | Santiago Esteban Contreras Aranda |
| Tipo de documento | DNI |
| Número de documento de identidad | 06445308 |
| **Miembro del jurado 1** | |
| Nombres y apellidos | Luis Javier Vásquez Serpa |
| Tipo de documento | DNI |
| Número de documento de identidad | 43389380 |
| **Miembro del jurado 2** | |
| Nombres y apellidos | Bertha Ulloa Rubio |
| Tipo de documento | DNI |
| Número de documento de identidad | 06445309 |
| **Datos de investigación** | |
| Línea de investigación | A.3.1.4. Matemática Aplicada |

| | |
|---|---|
| Grupo de investigación | CIENCIAS MATEMÁTICAS PARA LAS CIENCIAS DE LA VIDA - GI CMATVIDA |
| Agencia de financiamiento | Sin financiamiento |
| Ubicación geográfica de la investigación | Universidad Nacional Mayor de San Marcos<br>Edificio: Facultad de Ciencias Matemáticas<br>País: Perú<br>Departamento: Lima<br>Provincia: Lima<br>Distrito: Cercado de Lima<br>Manzana y lote: 3400<br>Calle: Av. Carlos Germán Amezaga<br>Latitud: -12.0602078<br>Longitud: -77.0820699 |
| Año o rango de años en que se realizó la investigación | Obligatorio. Ejemplo:<br>2019 - 2022 |
| URL de disciplinas OCDE | Matemáticas aplicadas<br>https://purl.org/pe-repo/ocde/ford#1.01.02<br>Informática y Ciencias de la Información<br>https://purl.org/pe-repo/ocde/ford#1.02.00 |

**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**
Universidad del Perú, DECANA DE AMÉRICA

**FACULTAD DE CIENCIAS MATEMÁTICAS**
**VICEDECANATO DE INVESTIGACIÓN Y POSGRADO**
**UNIDAD DE POSGRADO**

# ACTA DE SUSTENTACIÓN DE TESIS PARA OPTAR EL GRADO ACADÉMICO DE MAGÍSTER

Siendo las 14:00 horas del día miércoles veintiocho de junio de dos mil veintitrés, en el Laboratorio UPG de la Facultad de Ciencias Matemáticas, se reunió el Jurado Evaluador de tesis, presidido por el Dr. Santiago Esteban Contreras Aranda e integrado por los siguientes miembros: Mg. Luis Javier Vásquez Serpa (Jurado Informante) y la Dra. Roxana López Cruz como Asesora; y con la participación del Dra. Bertha Ulloa Rubio (Jurado Externo) en la sala virtual (meet.google.com/rdo-vjev-nox), para la sustentación de la tesis titulada: **«DINÁMICA DE PROYECTOS DE DESARROLLO DE SOFTWARE MULTICAPA A PARTIR DE LA EVOLUCIÓN CUANTITATIVA DEL CÓDIGO FUENTE»** presentada por el Señor **Bachiller DIEGO BRAVO ESTRADA»**, para optar el **Grado Académico de Magíster en Matemática Aplicada con mención en Matemática Computacional**.

Concluida la exposición, los miembros del Jurado de Tesis procedieron a formular sus preguntas que fueron absueltas por el graduando; acto seguido se procedió a la evaluación correspondiente, según tabla adjunta, habiendo obtenido el Señor **Bachiller DIEGO BRAVO ESTRADA**, el calificativo de 18 (MUY BUENO).

Habiendo sido aprobada la sustentación de la Tesis, el Jurado Evaluador recomienda para que el Consejo de Facultad apruebe el otorgamiento del **Grado Académico de Magíster en Matemática Aplicada con mención en Matemática Computacional** al **Bachiller DIEGO BRAVO ESTRADA.**

Siendo las 15:40 horas, se levantó la sesión, firmando para constancia la presente Acta:

Dr. Santiago Esteban Contreras Aranda
**PRESIDENTE**

Mg. Luis Javier Vásquez Serpa
**MIEMBRO**

Dra. Bertha Ulloa Rubio
**MIEMBRO EXTERNO**

Dra. Roxana López Cruz
**ASESORA**

**Universidad Nacional Mayor de San Marcos**

Universidad del Perú. Decana de América

**FACULTAD DE CIENCIAS MATEMÁTICAS**

**VICEDECANATO DE INVESTIGACIÓN Y POSGRADO**

**UNIDAD DE POSGRADO**

---

# Informe de Evaluación de Originalidad

Vicedecano de Investigación y Posgrado: Dr. Eugenio Cabanillas Lapa

Operador: Mag. José Luis Acuña Guillermo

Tesis: "DINÁMICA DE PROYECTOS DE DESARROLLO DE SOFTWARE MULTICAPA A PARTIR DE LA EVOLUCIÓN CUANTITATIVA DEL CÓDIGO FUENTE"

Tesista: DIEGO BRAVO ESTRADA

Recepción de documento: 17/03/2023

Aplicación del software: 17/03/2023

Software utilizado: Turnitin

Configuración del software:

- Excluir material bibliográfico
- Excluir material citado
- Excluir fuentes pequeñas N° palabras: 40
- Sin depósito

Resultado del análisis con el software: cuatro por ciento (4%)

Fuentes de originales de las similitudes encontradas:

- Ver abajo

Documento cumple con criterios de originalidad.

- **El documento cumple los criterios de originalidad. <u>Sin observaciones</u>.**

Fecha del informe: 18/03/2023

UNMSM

Firmado digitalmente por CABANILLAS LAPA Eugenio FAU 20148092282 soft
Motivo: Soy el autor del documento
Fecha: 17.03.2023 23:37:48 -05:00

**Dr. Eugenio Cabanillas Lapa**

**VICEDECANO DE INVESTIGACIÓN Y POSGRADO**

# Índice general


**1. Nociones Básicas** **11**

1.1. Situación Problemática . . . . . 11

1.1.1. Formulación del Problema . . . . . 13

1.1.2. Justificación de la Investigación . . . . . 14

1.2. Antecedentes de la investigación . . . . . 15

1.2.1. Modelos de Estimación de Esfuerzo . . . . . 15

1.2.1.1. Software Science . . . . . 17

1.2.1.2. Complejidad Ciclomática . . . . . 18

1.2.1.3. Complejidad Espacial . . . . . 18

1.2.1.4. Puntos de Función . . . . . 20

1.2.1.5. Análisis de Regresión Lineal . . . . . 22

1.2.1.6. COCOMO . . . . . 23

1.2.1.7. Modelo de Norden/Putnam . . . . . 25

1.2.1.8. Leyes de la Evolución del Software . . . . . 27

1.2.2. Acerca de los Modelos de Estimación de Esfuerzo . . . . . 28

1.2.3. Modelos de Fiabilidad . . . . . 35

1.2.3.1. El modelo de Goel-Okumoto . . . . . 35

1.2.3.2. Modelamiento estocástico de Cambios en Código Fuente . . . . . 37

**2. Bases Teóricas** **38**

2.1. Modelo de Regresión Lineal . . . . . 38

2.1.1. Convención Matricial de Derivación Parcial . . . . . 38

2.1.2. Sistema de Ecuaciones Lineales . . . . . 39

2.1.3. Minimización Cuadrática . . . . . 40

2.1.4. Varianza y Covarianza . . . . . 40

2.1.5. Asunciones del Modelo de Regresión Clásico . . . . . 42

2.1.6. Algunas Consecuencias del Modelo de Regresión Clásico . 42

2.1.7. Interpolación de Polinomios . . . . . 43

2.2. Distribución de Poisson y Procesos de Poisson . . . . . 44

2.2.1. Distribución Binomial . . . . . 44

2.2.2. Procesos de Conteo . . . . . 45
2.2.3. Proceso de Poisson . . . . . 46
2.2.4. Proceso de Poisson no Homogéneo (NHPP) . . . . . 48
2.3. Estimación de Parámetros para NHPP . . . . . 48
2.3.1. Estimación para modelo Goel-Okumoto . . . . . 50
2.3.2. Contrastación de Modelos Estadísticos . . . . . 51
2.4. Conceptos de Teoría de Fiabilidad . . . . . 52
2.4.1. Ocurrencia de fallos en el proceso de Poisson . . . . . 55
2.5. Promedio Móvil . . . . . 55
2.6. Técnicas de Optimización . . . . . 57
2.6.1. Problema general de Programación Lineal . . . . . 57
2.6.2. Formas Cuadráticas . . . . . 59
2.6.3. Condiciones para Extremos . . . . . 59
2.6.4. Sistemas de Ecuaciones No Lineales . . . . . 60
2.6.5. Optimización con restricciones en forma de igualdades[1] . . 61
2.6.6. Optimización con restricciones en forma de desigualdades . 65
2.6.7. Programación Cuadrática . . . . . 67
2.6.8. Programación Cuadrática Secuencial . . . . . 68

**3. Datos de Muestra y su Procesamiento 70**
3.1. Diseño de Investigación . . . . . 70
3.2. Población de Estudio . . . . . 71
3.2.1. Unidad de Análisis . . . . . 71
3.2.2. Tamaño y Selección de la muestra . . . . . 72
3.3. Técnicas de recolección de Datos . . . . . 74
3.3.1. Programas de extracción . . . . . 75
3.4. Procesamiento de datos . . . . . 75
3.4.1. Exploración Visual . . . . . 75
3.4.2. Escalas Temporales . . . . . 80
3.4.3. Preparación de los Datos . . . . . 83

**4. Propuesta de Modelamiento 87**
4.1. La Dinámica del Proyecto de Desarrollo de Software . . . . . 87
4.1.1. El Inicio del Proyecto . . . . . 87
4.1.2. Los recursos humanos asignados al proyecto . . . . . 89
4.1.3. Modelamiento Cuantitativo . . . . . 90
4.2. Modelamiento de la Tendencia Ideal . . . . . 93
4.2.1. Planteamiento Analítico de Tendencia Ideal . . . . . 94

[1]Esta sección y la subsiguiente han sido extraídas de la sección 7.4 de Foulds (1981) y algunas indicaciones de Nocedal y Wright (2006).

4.2.2. Caracterización de la función ELI . . . . . 97
4.2.2.1. Aproximación Heurística de la función ELI . . . . 98
4.2.2.2. Ajuste de ELI mediante Optimización . . . . . 100
4.2.3. Modelos Alternativos a ELI . . . . . 103
4.2.3.1. Modelo Logarítmico-Lineal Directo: LLD . . . . . 103
4.2.3.2. Modelo Exponencial-Lineal Directo: ELD . . . . 103
4.2.3.3. Modelos Polinomiales . . . . . 104
4.2.4. Ajuste de los Modelos Alternativos . . . . . 104
4.2.5. Contrastación Empírica de Modelos para Tendencia Ideal . 105
4.2.5.1. Métricas de Bondad de Ajuste . . . . . 105
4.2.5.2. Extracción de Datos para Tendencia Ideal . . . . 106
4.2.5.3. Resultados NRMSE . . . . . 107
4.2.5.4. Resultados MAE . . . . . 109
4.2.5.5. Resultados para la Productividad . . . . . 111
4.2.6. Contraste de Modelos de Tendencia Ideal . . . . . 112
4.2.6.1. Evaluación estadística de calidad de ajuste . . . . 112
4.2.6.2. Predicción de la Duración del Proyecto . . . . . 114
4.2.6.3. Estimación del tamaño final del código fuente . . 122
4.2.6.4. Relaciones de Productividad . . . . . 123
4.3. Modelamiento de la Tendencia Estocástica Procesual . . . . . 124
4.3.1. Planteamiento Analítico de Tendencia Estocástica . . . . . 124
4.3.2. Ajuste de Parámetros para Tendencia Estocástica . . . . . 126
4.3.2.1. Modelo Goel-Okumoto . . . . . 127
4.3.2.2. Modelo LLD . . . . . 127
4.3.2.3. Modelo ELD . . . . . 127
4.3.2.4. Modelo ELI . . . . . 128
4.3.3. Contraste Empírico de Modelos para Tendencia Estocástica 128
4.3.3.1. Métricas de Bondad de la estimación paramétrica 128
4.3.3.2. Resultados de la estimación paramétrica . . . . . 128
4.3.3.3. Resultados para Intensidad de Fallos . . . . . 129
4.3.3.4. Relación Tamaño-Picos . . . . . 131
4.3.3.5. Predicción vía tendencia estocástica . . . . . 131
4.4. Sobre la Consolidación de los Modelos . . . . . 132

# Índice de figuras

1.2.1. Curva de Norden para fuerza de trabajo. . . . . 25
1.2.2. Proyectos “a lo grande” vs. desarrollo “en pequeño”. . . . . 29

2.5.1. Índice Beveridge y Promedio Móvil. . . . . 56

3.4.1. Codificación a Escala Diaria en proyecto “RE” . . . . . 77
3.4.2. Codificación a Escala Semanal en proyecto “RE” . . . . . 78
3.4.3. Codificación a Escala Mensual en proyecto “RE” . . . . . 79
3.4.4. Gráfico acumulativo de código fuente neto en proyecto “RE”. . . . 79
3.4.5. Codificación a Escala Semanal en proyecto “SP” . . . . . 80
3.4.6. Codificación a Escala Semanal en proyecto “AD” . . . . . 80
3.4.7. Codificación a Escala Semanal en proyecto “AS” . . . . . 81
3.4.8. Codificación a Escala Semanal en proyecto “JP” . . . . . 81
3.4.9. Codificación a Escala Semanal en proyecto “CT” . . . . . 81
3.4.10. Codificación a Escala Semanal en proyecto “PO” . . . . . 82
3.4.11. Gráfico de Frecuencia proyecto “SP” (L-D.) . . . . . 83
3.4.12. Gráfico de Frecuencia proyecto “SP” (L-V.) . . . . . 84
3.4.13. Gráfico de Frecuencia proyecto “RE” (L-D.) . . . . . 84
3.4.14. Gráfico de Frecuencia proyecto “CT” (L-D.) . . . . . 85
3.4.15. LOC diario original proyecto “RE”. . . . . 85
3.4.16. LOC diario consolidado proyecto “RE”. . . . . 86

4.2.1. Modelo ELI para Tendencia Ideal $N(t)$. . . . . 101
4.2.2. Modelo ELI para Productividad Ideal $N'(t)$. . . . . 102
4.2.3. LOC semanal proyecto “AD”. . . . . 107
4.2.4. Ajuste proyecto “AD” con modelo cúbico. . . . . 108
4.2.5. Ajuste proyecto “AD” con modelo ELI. . . . . 108
4.2.6. Ajuste proyecto “SP” con modelo ELI. . . . . 109
4.2.7. Ajuste proyecto “SP” con modelo LLD. . . . . 110
4.2.8. Predicción AD via tendencia ideal. . . . . 116
4.2.9. Predicción SP via tendencia ideal desde 4 semanas. . . . . 116
4.2.10. Predicción SP via tendencia ideal desde 8 semanas. . . . . 117

4.2.11 Predicción RE via tendencia ideal desde 4 semanas. . . . . 117
4.2.12 Predicción RE via tendencia ideal desde 8 semanas. . . . . 118
4.2.13 Predicción AS via tendencia ideal desde 4 semanas. . . . . 118
4.2.14 Predicción AS via tendencia ideal desde 8 semanas. . . . . 119
4.2.15 Predicción JP via tendencia ideal desde 4 semanas. . . . . 119
4.2.16 Predicción CT via tendencia ideal desde 4 semanas. . . . . 120
4.2.17 Predicción CT via tendencia ideal desde 8 semanas. . . . . 120
4.2.18 Predicción PO via tendencia ideal desde 4 semanas. . . . . 121
4.2.19 Predicción PO via tendencia ideal desde 8 semanas. . . . . 121
4.2.20 Productividad ELI final de proyectos y predicción. Productividad ELI en color morado; predicción según modelo de la ecuación 4.2.18 en amarillo. . . . . 123
4.3.1. Predicción de tendencia estocástica proyecto SP . . . . . 133
4.3.2. Predicción de tendencia estocástica proyecto RE . . . . . 133
4.3.3. Predicción de tendencia estocástica proyecto CT . . . . . 134
4.3.4. Predicción de tendencia estocástica proyecto PO . . . . . 134

# Resumen

Se propone un modelo matemático/estadístico para describir la evolución del volumen de código fuente de un proyecto de desarrollo de software, a lo largo del tiempo. Este modelo se contrasta con información histórica de proyectos de desarrollo reales que han estado a cargo del autor.

La implementación del modelo pretende ser una herramienta de planificación, evaluación y toma de decisión para las organizaciones que tienen a cargo la ejecución de proyectos de desarrollo con las características que aquí se consideran.

## Palabras Clave

Evolución de Proyectos, Estimación de esfuerzo, Dimensionamiento de software, Desarrollo de Software, Ingeniería de Software, Costeo de Proyectos.

# Abstract

A mathematical/statistical model is proposed in order to describe the source code volume time evolution for a software development project. This model is contrasted with historical data provided by real development projects which were managed by the author.

The model implementation aims to be a planning, evaluation and decision making tool for organizations involved in software development projects.

## Keywords

Project evolution, Effort estimation, Software sizing, Software development, Software Engineering, Project cost management.

# Glosario

## Análisis de Regresión

"El análisis de regresión trata del estudio de la dependencia de una variable (variable dependiente) respecto de una o más variables (variables explicativas) con el objetivo de estimar o predecir la media o valor promedio poblacional de la primera en términos de los valores conocidos o fijos (en muestras repetidas) de las segundas" (Gujarati y Porter, 2010, p.15).

## Arquitectura Multicapa

Estructura de un aplicativo o aplicativos de computador en el cual las funciones de presentación, de procesamiento y/o lógica de negocio, y la administración de los datos están lógica y/o físicamente separadas en compartimentos o contenedores. Consultar la sección "What Is a Layered Architecture?" en Brown y col. (2003) para más detalles.

## Artefactos de Software

Corresponde a los productos y subproductos del proceso de desarrollo de software. Normalmente materializados en archivos de computador de tipo y propósito diverso que constituyen parte de un aplicativo o contienen información necesaria para su ejecución. En StackExchange (2011) se proporcionan diversas visiones de este concepto.

## Base de Datos Relacional

En el modelo relacional los datos son organizados en relaciones o tablas, en que una tupla o fila representa una ocurrencia de un fenómeno, y una columna representa un atributo de este fenómeno. Una relación debe tener uno o más atributos, cuyos valores identifican unívocamente a las tuplas. Todas las relaciones entre distintas relaciones se representan explícitamente como valores de atributos. Por tanto, los datos de distintas relaciones pueden ser conectados encajando

valores de los atributos correspondientes. Para la manipulación de las relaciones se ha definido un conjunto de operadores (álgebra relacional.) Los operadores trabajan en una base orientada a conjuntos: esto es, ellos consideran las relaciones-operandos como conjuntos de tuplas y la relaciones-resultado es un conjunto de tuplas. Existen algoritmos bien establecidos para el diseño de un esquema de base de datos relacional.(Morris y Tamm, 1993, p.94)

## Ciclo de Vida

El ciclo de vide del software cubre el periodo temporal desde el planteamiento de un problema a ser resuelto por un sistema hasta la eliminación del sistema en uso. Múltiples modelos de ciclo de vida, basados en distintas aproximaciones y paradigmas, han sido propuestos.(Morris y Tamm, 1993, p.191)

## Framework

Un framework es un conjunto de clases que cooperan y que constituyen un diseño reutilizable para una clase específica de software. [...] El framework dicta la arquitectura de la aplicación. Definirá la estructura general, su particionamiento en clases y objetos, sus responsabilidades principales, cómo colaboran las clases y objetos, y el hilo de control. Un framework predefine los parámetros de diseño; el diseñador/implementador puede concentrarse en los aspectos específicos de la aplicación. El framework captura las decisiones de diseño que son comunes al dominio de la aplicación. (Gamma y col., 1994)

## Interfaz Web

En este contexto significa que el software desarrollado proporciona acceso al usuario mediante el uso de un “navegador” o “web browser”, a través de las redes de computador estándar.

## Lenguaje 3GL

Lenguaje de tercera generación: En programación, un lenguaje avanzado de alto nivel que proporciona facilidades tales como abstracción de datos y control asíncrono de procesos. Modula 2 y Ada son lenguajes de tercera generación típicos. (Longley, Shain y Eds, 1985, p.505)

## Lenguaje de programación Java

"Java es, por tanto, muy valioso para entornos de red distribuida como el Web, pero rebasa con mucho ese campo y constituye un potente lenguaje de programación de uso general, adecuado para crear una amplia gama de aplicaciones que no dependen de las características de red o que las necesitan por diferentes razones. La capacidad de Java para ejecutar código en hosts remotos de manera segura es requisito esencial para muchas organizaciones.

Otros grupos utilizan Java como lenguaje de programación de uso general adecuado para proyectos en los que la independencia de la máquina es menos importante. La facilidad de programación de Java y sus características de seguridad ayudan a producir rápidamente código depurado. Algunos errores habituales de programación nunca llegan a producirse debido a características como la recogida de basura y las referencias seguras con respecto al tipo. Las modernas aplicaciones basadas en red y en interfaz gráfica de usuario que deben atender simultáneamente diversas tareas quedan abastecidas por el soporte de Java para multithreading mientras que los mecanismos de manejo de excepciones facilitan la tarea de resolver situaciones de error. Aunque las herramientas que incorpora son potentes, Java es en sí mismo un lenguaje sencillo que los programadores pueden dominar rápidamente." (Arnold y Gosling, 1997, p.xiii)

## Métrica de Software

El término "métricas de software" es usado como una etiqueta genérica que cubre todos los aspectos de cuantificación relacionados a la producción y soporte de software. Puede considerarse una definición adecuada si se acuerda que el término "métrica" incluye valores obtenidos en al menos tres formas: Por edicto (valores fijados que condicionan el proyecto); por estimación, cuando el valor no está disponible pero se obtiene mediante modelos o propuestas subjetivas; finalmente, mediante medición, cuando el valor puede ser obtenido directamente. (B. A. Kitchenham, 1990, p.153)

## Modelo ELI

El modelo de "tendencia ideal" que se propone en este trabajo; las iniciales ELI corresponden a "Exponencial Lineal Inversa".

## Octave

GNU Octave es un lenguaje de alto nivel, primordialmente orientado a cálculos numéricos. Proporciona una interfaz de línea de comando conveniente para

resolver problemas lineales y no lineales numéricamente, y para ejecutar otros experimentos numéricos usando un lenguaje que es mayormente compatible con Matlab. También puede ser usada como un lenguaje orientado a lotes. (Eaton, 2018)

# PHP

PHP [...] es un lenguaje de guión de propósito general de código abierto ampliamente utilizado que es especialmente adecuado para desarrollo web y puede ser embutido en HTML. [...] Lo que distingue a PHP de algo como JavaScript del lado del cliente es que código es ejecutado en el servidor, generando HTML el cual es enviado al cliente. (PHP PHP Group, 2018)

## Proyecto de Software

Iniciativa temporal ejecutada para crear un único producto, servicio o resultado. [...] Los proyectos de software con frecuencia se emprenden para modificar un producto de software existente, para integrar un conjunto de componentes de software existentes, para extender las capacidades de los productos de software, o para modificar la infraestructura de software de la organización. (PMI, 2013, p.4)

## Prototipo

Un prototipo de software es un modelo que permite a los ingenieros de software probar el comportamiento de un sistema en las etapas tempranas del desarrollo de software. Muchos prototipos de software también permiten a los usuarios finales del sistema interactuar con el comportamiento propuesto del sistema y evaluar qué tan bien satisface sus necesidades, antes de que el sistema sea completado. (Morris y Tamm, 1993, p.273)

## RAD

"RAD [Rapid-Application Development] es un conjunto integrado de técnicas, lineamientos y herramientas que facilitan el despliegue de las necesidades de software del cliente en un periodo de tiempo corto."(Gottesdiener, 1995) Pone énfasis en el uso de prototipos, reconociendo que el desarrollo de software es un proceso de continuo aprendizaje, el cual proporciona experiencia con la que el desarrollo puede ser retroalimentado.

## Shell

"El intérprete de mandatos o "shell" es la interfaz principal entre el usuario y el sistema, permitiéndole a aquél interactuar con los recursos de éste. El usuario introduce sus órdenes, el intérprete las procesa y genera la salida correspondiente."(Gómez, 2005, p.5)

## Subversion

Subversion es un sistema centralizado para compartir información. En su núcleo, es un repositorio, que es un almacen central de datos. El repositorio almacena información en la forma de un sistema de archivos en árbol - una jerarquía típica de archivos y directorios. Cualquier número de clientes se conectan al repositorio y entonces leen y escriben esos archivos. Al escribir datos, el cliente hace disponible esta información a otros; al leer los datos, el cliente recoge información de los otros. (Pilato, Collins-Sussman y Fitzpatrick, 2008, p.1)

# INTRODUCCIÓN

Este estudio fue motivado por un problema clásico al que se enfrentan todos los gestores e interesados en el desarrollo de software: ¿cuánto tiempo requiere la construcción de un programa o aplicativo destinado a cumplir ciertas tareas? En términos más formales, este problema consiste en la estimación de los recursos (personal, tiempo, infraestructura, costos, etc.) necesarios para la implementación por medio de software de un conjunto de requerimientos, típicamente en el marco de un emprendimiento planificado popularmente conocido como "proyecto".

En nuestra experiencia profesional, hemos sido testigo de la extrema informalidad con la que este problema es usualmente abordado. Tal como detallaremos más adelante, el problema de la estimación de esfuerzo de proyectos de software parecería ser insoluble en su expresión más general; sin embargo, bajo una serie de supuestos complementarios (cuya razonabilidad siempre es materia de discusión), los investigadores de la ingeniería de software han presentado diversas respuestas aproximadas, pero de cuestionable utilidad.

Un segundo asunto relacionado con lo anterior corresponde al control del avance de los proyectos de software; aquí se trata de proyectos de software cuya codificación ya se ha iniciado, y cuyo producto en evolución se debe monitorear a fin de absolver preguntas usuales tales como ¿el proyecto está cumpliendo las metas programadas a su tiempo? ¿el proyecto se está retrasando? ¿el proyecto acabará a tiempo? ¿los productos resultantes (si bien parciales), son satisfactorios? etc.

A diferencia del escenario anterior, aquí se cuenta con cierta experiencia preliminar en cuanto a los resultados que están siendo obtenidos a través del tiempo; es decir, disponemos de información acerca de la "dinámica" de generación del código fuente. Un camino razonable para el aprovechamiento de esta información consiste en extrapolar esta dinámica, con miras a obtener predicciones útiles para el control del proyecto. Es en este orden de ideas que hemos planteado la presente investigación.

Desde un punto de vista más personal, un incentivo importante para considerar el estudio de la dinámica de la codificación radica en la disponibilidad de información histórica de proyectos que fueron gestionados por el autor, la misma que ha sido acumulada a lo largo de los últimos diez años.

En suma, el objetivo general consiste en establecer un modelo cuantitativo que refleje objetivamente la dinámica temporal del código fuente para proyectos de desarrollo de software en un contexto tecnológico y organizacional determinado, con miras a obtener indicadores útiles para el control exitoso de aquéllos.

En cuanto al contenido de este documento, el primer capítulo brinda una revisión histórica referente a las diversas alternativas planteadas al problema de estimación de proyectos (algunas de las cuales están relacionadas a su dinámica); la literatura al respecto es abundante, por lo que nuestra selección estará sesgada hacia aquellos planteamientos basados en modelos matemáticos (y estadísticos) que están más relacionados a nuestra propuesta. En particular, hemos obviado aquellos métodos basados en técnicas de inteligencia artificial que se han popularizado en la última década.

El segundo capítulo proporciona una revisión de un conjunto de técnicas matemáticas y estadísticas que serán empleadas en nuestra propuesta. A continuación, el tercer capítulo proporciona una breve revisión acerca de los datos (provenientes de proyectos reales) que utilizaremos para contrastar nuestros modelos (unidad de muestra) y que en buena parte han guiado la propuesta; esta última es presentada en el cuarto capítulo. Como se verá, la propuesta viene dada en dos frentes: tendencia ideal (determinista) y tendencia estocástica, los que reflejan -respectivamente- el patrón general que gobierna la dinámica del desarrollo del software, y los eventos "inesperados" propios de las actividades con fuerte intervención humana. Al final de este mismo capítulo se presentan algunas conclusiones y posibles líneas de investigación últerior.

# Capítulo 1

# Nociones Básicas

## 1.1. Situación Problemática

Los proyectos de desarrollo de software constituyen una actividad de importancia primordial en el mundo actual dada la ubicuidad de la informática en todos los aspectos de la sociedad moderna. La "dinámica" de los proyectos, entendida como ejecución de actividades a lo largo del tiempo, es ampliamente estudiada desde el punto de vista metodológico en la "ingeniería de software" y disciplinas afines. Entender esta dinámica es importante para la gestión de las organizaciones dada la preeminencia que tiene el proyecto como mecanismo general de consecución de objetivos; en particular, es relevante para la estimación objetiva de "plazos comprometidos" y el empleo eficiente de los recursos humanos:

> Entregar los productos de software a tiempo, dentro del presupuesto, y con un nivel de calidad acordado, es un asunto crítico para muchas organizaciones de software. Subestimar los costos del software puede tener efectos perjudiciales en la calidad del mismo, y por tanto en la reputación del negocio. De otro lado, sobreestimar el costo del software puede resultar en oportunidades perdidas para financiar otros proyectos.(R. Jeffery, Ruhe y Wieczorek, 2000, p.1009)

El estudio sistemático del diseño e implementación del software (en particular, mediante proyectos de desarrollo) corresponde al campo conocido como "ingeniería de software"[1]. Dentro de la planificación de proyectos, uno de sus problemas tradicionales constituye el de la "estimación de esfuerzo", el cual comprende la definición de los recursos humanos y la duración necesarios para la consecución del producto de software.

[1]ISO/IEC 2382 define por ingeniería de software: Aplicación sistemática de conocimiento científico y tecnológico, modelos y experiencia al diseño, implementación, pruebas y documentación de software para optimizar su producción, soporte y calidad (ISO, 2005).

La mala estimación de los recursos que requiere un proyecto es una causa común para el fracaso de los proyectos de software (Charette, 2005)[2]; sin embargo, es bien conocida la dificultad de efectuar estimaciones en éstos (Stutzke, 2005, p.18). Según Galorath y Evans (2006, p.14) las principales causas de error (en las estimaciones) corresponden a un mal dimensionamiento del software a ser construido, así como a:

- Carencia o mala utilización de datos históricos
- Gerencia o liderazgo excesivamente optimista
- El no hacer uso de las estimaciones
- El no mantener la estimación actualizada

En esa línea, PMI señala en lo referente a la planificación de los costos de los proyectos de software:

> La estimación de software es propensa a error; predecir la precisión de un estimado es difícil debido a los muchos factores que pueden afectar la estimación. Los valores de muchos de esos factores no son conocidos durante la planificación inicial. Un grueso estimado del orden de magnitud se genera típicamente durante la fase de inicio de un proyecto de software, cuando los requerimientos son inmaduros, los parámetros reales del desarrollo del software están siendo formulados, y el equipo de desarrollo puede o no haber sido identificado. (PMI, 2013)

Si bien es ampliamente aceptado que las facetas no racionales del ser humano tienen un considerable efecto en la planificación y ejecución de los proyectos, son muy raras las propuestas que las incorporan en el análisis[3]. Desde este punto de vista, todas estas propuestas estarían condenadas al fracaso. Esto no ha impedido el planteamiento de numerosos métodos y modelos cuantitativos para predecir las variables mencionadas (métodos y modelos de estimación de esfuerzo) los cuales -en opinión de diversos autores- han tenido una utilidad práctica muy limitada (Brown y col., 2003; Heemstra, 1992). Según Briand, Basili y Thomas (1993, p.152) la construcción de modelos empíricos en ingeniería de software se enfrenta a los siguientes problemas:

[2]Los primeros capítulos de Stepanek (2005) proporcionan una explicación sistemática acerca de los motivos por los que los proyectos de software fracasan con mayor frecuencia que los de las industrias tradicionales.

[3]Para un ejemplo donde se considera un "modelo político" (informal) como complemento al "modelo racional" (formal), consultar Lederer y col. (1990).

- No hay una teoría probadamente efectiva en algún entorno que proporcione una relación formal entre las métricas registradas en el desarrollo del proyecto
- Hay muy poca evidencia que sustente las densidades de probabilidad asumidas para las variables de interés
- El tamaño de la muestra suele ser muy reducido en relación a las exigencias de las técnicas clásicas de la estadística
- Los modelos deben tratar con variables ausentes, interdependientes y no relevantes, debido a la falta de comprensión del proceso de desarrollo de software
- El registro de datos en ambientes de producción es una tarea difícil

Como veremos más adelante, otro problema muy serio corresponde a la validez de la "data" reportada. Es por todo esto que algunos investigadores reportan la aparente imposibilidad de confeccionar un único modelo que logre ser exitoso para la planificación de todo tipo de proyecto de software, recomendándose la elaboración de modelos específicos para contextos de desarrollo concretos (un tipo de producto específico, un único lenguaje de programación, una única organización, etc.)

A tal efecto, nuestro trabajo es una propuesta dirigida a una clase específica de proyectos y tecnologías actualmente muy utilizadas, así como un patrón organizacional definido. Se pretende describir la dinámica de tales proyectos a partir de información detallada proveniente de proyectos reales, específicamente, de sus cambios cuantitativos en el código fuente. Pese a su relevancia, en la presente investigación no se considera el análisis de los recursos humanos desde el punto de vista de sus competencias, metas personales, ambiente de trabajo, rasgos psicológicos, etcétera; todo lo cual consideramos que puede ser materia de estudios ulteriores.

### 1.1.1. Formulación del Problema

El problema se formula del siguiente modo: dado un conjunto de requerimientos identificados como susceptibles de ser implementados mediante un proyecto de desarrollo de software de computador, ¿cómo la Dinámica de Proyectos de Desarrollo de Software Multicapa evoluciona cuantitativamente a partir del Código Fuente?

A partir de este problema general surgen preguntas más específicas tales como:

- ¿Qué relación existe entre esta dinámica y el esfuerzo (en horas-persona) involucrado?
- ¿Cómo puede emplearse esta información para efectos de controlar objetivamente el "avance" del proyecto?
- ¿Qué relación existe entre la dinámica del código fuente y la calidad del producto resultante?

Esto corresponde a un conjunto de problemas específicos derivados:

P1 ¿Cómo la Dinámica de Proyectos de Desarrollo de Software Multicapa cumple con la evolución cuantitativamente de la estimación de esfuerzo del personal, a partir del Código Fuente del Software?

P2 ¿Cómo la Dinámica de Proyectos de Desarrollo de Software Multicapa cumple con la evolución cuantitativamente del tiempo de desarrollo del algoritmo, a partir del Código Fuente del Software?

P3 ¿Cómo la Dinámica de Proyectos de Desarrollo de Software Multicapa cumple con la evolución cuantitativamente de la complejidad de su infraestructura a partir del Código Fuente del Software?

P4 ¿Cómo la Dinámica de Proyectos de Desarrollo de Software Multicapa cumple con la evolución cuantitativamente de la funcionalidad del costo, a partir del Código Fuente del Software?

El alcance del presente trabajo corresponde al planteamiento de un modelo matemático/estadístico de la dinámica cuantitativa del código fuente, tomando como punto de partida la información histórica de un conjunto de proyectos de desarrollo de software gestionados por el autor en años anteriores.

Estos proyectos de desarrollo están restringidos al escenario de arquitectura multicapa con interfaz web y almacenamiento relacional, teniendo como propósito la automatización de procesos operativos del usuario solicitante (en particular, no incluimos portales institucionales o publicitarios de cara a Internet.)

Anotemos que estas restricciones nos dejan con un segmento extremadamente amplio de proyectos de desarrollo de software que se vienen desarrollando y utilizando en la actualidad, y cuya importancia es de primer orden para nuestra sociedad.

### 1.1.2. Justificación de la Investigación

El conocimiento de la dinámica cuantitativa del código fuente proporciona información objetiva en relación a la marcha de un proyecto y su posible conclusión. Esta información es muy valiosa pues las organizaciones requieren evaluar

continuamente el cumplimiento de la planificación a fin de garantizar la consecución de las metas comprometidas, así como controlar los costos involucrados. Esta evaluación actualmente se suele llevar a cabo mediante el control de "hitos" de planificación, lo cual suele adolecer de una gran subjetividad.

La utilidad de este estudio es aplicable a toda organización pública o privada que emprende un proyecto de desarrollo de software (con las características anteriormente señaladas) a fin de controlar su avance y su rentabilidad.

Desde un punto de vista académico, esta investigación se asocia al perfil de la Maestría en Matemática Aplicada con mención en Matemática Computacional, en tanto propone la elaboración de un modelo matemático para un problema real haciendo uso de técnicas computacionales (UNMSM, 2016).

## 1.2. Antecedentes de la investigación

Nuestros antecedentes han sido organizados en dos grupos principales: modelos de estimación de esfuerzo en proyectos de software, y modelos de fiabilidad para productos de software.

### 1.2.1. Modelos de Estimación de Esfuerzo

Desde fines de los años 60 en que formalmente nace la "ingeniería de software", el problema de la planificación de los proyectos ha sido una de las preocupaciones más importantes de la misma; así, no es de extrañar que a lo largo de los años haya sido abordado desde diversos ángulos en numerosas publicaciones; en particular, en lo referido a la estimación de los recursos humanos necesarios, los plazos de ejecución, y los costos derivados.

En los últimos años la planificación de proyectos (de toda clase) se ha encuadrado en lo que podemos denominar la "subdisciplina" de gestión de proyectos[4], la cual tiene bajo su responsabilidad concentrar y sintetizar la información proveniente de diversos especialistas y actores relevantes.

Para el caso de proyectos de desarrollo de software, es usual considerar la "opinión" de los analistas y programadores que estarán a cargo de la ejecución de los mismos; esta opinión puede ser aquilatada de modo informal, o ulteriormente procesada mediante la aplicación de una amplia gama de métodos formales, los que suelen estar basados en algún modelo matemático y/o estadístico.

Cuando los estimados son generados esencialmente a partir de las opiniones (informadas) de los especialistas, se dice que el método empleado ha sido "basado

[4]La referencia usual es PMI (2013). Un enfoque alternativo (a nuestro criterio, más útil) se puede consultar en los primeros capítulos de Stepanek (2005).

en el experto". Los métodos "basados en el experto" han sido y son los más empleados de acuerdo a la mayoría de investigadores.(Molokken y Jorgensen, 2003; Arnuphaptrairong, 2018)

La alternativa a éstos corresponde al uso de "herramientas de estimación", las cuales operan como complemento a las herramientas generales de planificación de proyectos; según Capers Jones (2004), el uso de estas herramientas de estimación es un factor clave para el éxito de proyectos de gran extensión. Las herramientas de estimación utilizan diversas métodos para obtener sus resultados, pero éstos típicamente se concentran alrededor de un modelo matemático y/o estadístico[5].

Los métodos y modelos de estimación de esfuerzo son muy variados. En Trendowicz y R. Jeffery (2014) éstos son clasificados en "orientados a los datos" (data-driven), "basados en el experto" (expert-based) e híbridos; los modelos orientados a los datos se subclasifican en propietarios y no propietarios; los últimos nuevamente se clasifican en "basados en modelos", "basados en analogía" (contraste con la experiencia de proyectos previos y similares) y aquellos que combinan las dos últimas clases. Finalmente, los métodos basados en modelos son también clasificados en paramétricos, no paramétricos y semi-paramétricos. Para una discusión acerca de las clasificaciones empleadas en la literatura, consultar Vera, Ochoa y Perovich (2017).

En relación a los principales métodos en uso, existen diversas síntesis que dan cuenta del "estado del arte" (por ejemplo Osmanbegović, Suljić y Agić 2017; Wen y col. 2012; Keshta 2017), algunos con un enfoque hacia la puesta en práctica de los mismos (por ejemplo McConnell (2006), Laird y Brennan (2006), Bramble y col. (2003), Abran (2015) y Chemuturi (2009)). En las siguientes secciones haremos una breve revisión de algunos antecedentes, con énfasis en aquellos que están orientados hacia el modelamiento matemático.

Algunos autores han restringido el ámbito de estudio a proyectos ligados a un determinado contexto tecnológico. Por ejemplo, Mendes (2007) y Mendes (2014) describen extensamente las diferencias de diversa índole entre los proyectos de "desarrollo web" frente a los proyectos "convencionales" de cara a la estimación de esfuerzo. Sin embargo, estos sistemas de tipo web comprenden a su vez numerosas subcategorías en constante evolución; por ejemplo, Folgieri y col. (2011) proporciona una clasificación para los proyectos web basada en el nivel de automatización proporcionada por las herramientas de desarrollo, que en este contexto son abundantes[6]. Es claro que el esfuerzo (y el nivel de especialización requeri-

[5]Como es de esperarse, en ciertos productos se emplean métodos propietarios que no están claramente documentados. Estos casos no serán considerados en este trabajo.

[6]De acuerdo a este autor, el proyecto "estándar" no incorpora un "framework" ni emplea herramientas de tipo RAD; en el tipo "Standard RAD", el "esqueleto" es generado por una herramienta RAD; el tipo "Extreme RAD" sólo demanda un conocimiento de la estructura de

do) varía considerablemente de un tipo a otro; por ejemplo, los desarrollos de portales publicitarios suelen demandar un esfuerzo muy elevado en actividades de diseño gráfico, en comparación con la codificación del software asociado (este último podría no existir en el caso de portales "estáticos".)

#### 1.2.1.1. Software Science

A lo largo de la década de los setenta, Halstead (Halstead y col., 1977) elaboró un conjunto de indicadores con la finalidad de "medir" el código fuente de los programas. Esta medición intentaba proporcionar información más significativa que el conteo de líneas de código fuente.

A tal efecto propuso una serie de métricas obtenidas a partir de las unidades individuales del código fuente (análisis lexicográfico) las cuales correlacionó con un conjunto de proyectos a los que tuvo acceso. En particular estableció los siguientes indicadores:

**Volumen:**

$$V = N.log_2(\eta)$$

donde $\eta = \eta_1 + \eta_2$ es una medida del "vocabulario" del programa definida como la suma de los distintos operadores ($\eta_1$) y operandos ($\eta_2$) utilizados, mientras que $N = N_1 + N_2$ es la longitud del programa medida en número total (contados con repetición) de operadores ($N_1$) y operandos ($N_2$.)

**Nivel:**

$$L = (\frac{1 + M}{n_1})(\frac{n_2}{N_2})$$

donde $M$ es el número de módulos que constituyen el programa.

Estableciendo experimentalmente que el producto $V* = LxV$ es aproximadamente invariante.

Asimismo, mediante un (forzado) análisis del modelo mental del programador frente a una visión simplificada de lo que involucra implementar un algoritmo, propuso la siguiente fórmula para el número de "discriminaciones mentales" necesarias:

$$E = V/L = V^2/V*$$

Finalmente, este valor es igualado a los "momentos psicológicos" propuestos en algunas investigaciones de la psicología para establecer:

---

base de datos, y el flujo de trabajo (todo el resto es generado automáticamente.) Finalmente, el tipo "portal" orientado a la presentación de contenido, y por el contrario sus componentes de procesamiento de datos son muy limitados.

$$T = \frac{V^2}{SV*}$$

donde $T$ es el tiempo necesario para el desarrollo del algoritmo y $S$ es el "ratio" de "momentos" del cerebro por unidad de tiempo, obtenido experimentalmente.

No obstante el evidente atractivo y el interés inicial recibido, esta propuesta nunca cumplió con sus metas. En Coulter (1983) se puede consultar una crítica a su sospechosa aplicación de resultados de la psicología, mientras que Shen, Conte y Dunsmore (1983) y Shepperd e Ince (1994) resumen diversas críticas inherentes a la construcción cuantitativa de la teoría, así como su constatación experimental.

#### 1.2.1.2. Complejidad Ciclomática

Desarrollada por McCabe (McCabe, 1976), esta medida intentó cuantificar la complejidad de un programa o algoritmo mediante el análisis de las posibles rutas y/o ciclos de ejecución. En ese sentido el programa se modela como un grafo $G$ que representa el flujo que éste recorre. La complejidad está dada por el "número ciclomático":

$$V(G) = e - n + p$$

donde $n$ representa el número de vértices, $e$ el número de aristas y $p$ el número de componentes conectados.

El autor propuso diversas equivalencias de las secuencias de control típicas de los programas (notablemente en Fortran) a fin de automatizar el cálculo de la complejidad. Esta medida se asocia con frecuencia a las tareas de aseguramiento de calidad en la medida que (al menos en teoría) cada posible flujo del programa cuenta como un caso de prueba a ser verificado.

Numerosos autores han propuesto sendas adecuaciones a la complejidad ciclomática, así como diversas extensiones para ampliar su margen de aplicación (por ejemplo, existen diversas propuestas dirigidas a lenguajes orientados a objetos.)

El ya mencionado artículo de Shepperd e Ince (1994) también proporciona diversas críticas a la conceptualización y utilidad de la complejidad ciclomática.

#### 1.2.1.3. Complejidad Espacial

Lejanamente emparentadas con la anterior, las métricas de "complejidad espacial", y en particular la "code spatial complexity" [complejidad espacial del código fuente] (CSC), está asociada a la capacidad que tiene el desarrollador de comprender el programa:

> La complejidad de McCabe es una medida del control de flujo; las

> métricas de la ciencia [del software] de Halstead se concentran en el tamaño del software, promedio de variables por sentencia y debilidades en los programas. Pero esas clases de métricas no indican la complejidad relacionada al entendimiento y legibilidad del software. (Chhabra, Aggarwal y Singh, 2003, p.539)

Se define mediante:

$$CSC = \frac{\sum_{i=1}^{m} MCSC_i}{m}$$

donde $m$ es la cantidad de módulos de la aplicación, mientras que la complejidad espacial de un módulo se define por:

$$MCSC = \frac{\sum_{i=1}^{n} Dist_i}{n}$$

donde $n$ representa el conteo de invocaciones de ese módulo y $Dist_i$ es el valor absoluto de la diferencia en número de líneas entre la definición del módulo y el punto de invocación.

Esta complejidad varía dependiendo de la organización interna de los módulos constituyentes, pero observamos que es (aproximadamente) directamente proporcional al total de las líneas de código fuente (ver la figura 1 en la referencia citada.)

La complejidad espacial es analizada relación a tareas de mantenimiento de software, observándose nuevamente una relación de proporcionalidad directa entre el CSC y el tiempo de análisis necesario para proponer una determinada intervención en el programa. Esto significa que el CSC es inversamente proporcional a la productividad de estas tareas[7].

En nuestra opinión las medidas de complejidad de software (de cualquier clase que sean) ignoran un aspecto esencial del proceso de desarrollo: es muy frecuente invertir un gran esfuerzo en la interpretación y "traducción" de los requerimientos del proyecto a las estructuras computacionales del lenguaje de programación en uso, cuyo resultado puede lucir bastante sencillo en el código fuente; esto es muy deseable, y ocurre con frecuencia cuando se identifican adecuadamente las mejores estrategias de abstracción del problema en cuestión. Inversamente, un problema esencialmente sencillo (que en general, involucra menor esfuerzo de programación), puede tornarse en un código fuente muy confuso y/o extenso (que suele percibirse como de alta complejidad) cuando el análisis no es el adecuado, o

[7]El enfoque de la complejidad desde la perspectiva de la actividad cerebral involucrada (en contraste con los patrones del código fuente) se conoce colectivamente como "complejidad cognitiva" (Cant, D. R. Jeffery y Henderson-Sellers, 1995). La complejidad espacial pertenece a esta categoría.

(cosa frecuente) cuando el análisis inicial es incompleto y posteriormente deviene en una adaptación enrevesada.

#### 1.2.1.4. Puntos de Función

El análisis de puntos de función (Albrecht, 1979) proporciona un indicador que cuantifica la "funcionalidad" que brinda un programa con énfasis en la perspectiva del usuario, y se obtiene sumando diversos constituyentes ponderados por su respectiva complejidad.

Existen diversas variantes y extensiones para el análisis de puntos de función promovidas por grupos de usuarios y organizaciones[8]; la más popular corresponde al estándar del IFPUG (International Function Point Users Group) cuyas ideas principales enunciamos a continuación (consultar IFPUG (2010) para más información.)

El cálculo de los puntos de función IFPUG correspondiente a un componente de software o aplicativo, se inicia con el cálculo de un valor "no ajustado" para los puntos de función ($UFP$ = unadjusted function points):

$$UFP = (c_1 \times I) + (c_2 \times O) + (c_3 \times Q) + (c_4 \times F) + (c_5 \times E)$$

donde $I$ es el número de "entradas" desde el usuario, $O$ el número de "salidas" hacia el usuario, $Q$ el número de procesos de interacción con el usuario, $F$ el número de archivos principales (maestros) en el sistema, y $E$ el número de interfaces con otros sistemas. Los valores $c_i$ son ajustes de complejidad que se se encuentran en tablas y son obtenidos experimentalmente.

A continuación se debe analizar un conjunto de factores a los que podría estar sometido el software en cuestión y que potencialmente lo complejizan; por ejemplo, exigencias de alta performance, múltiples emplazamientos, capacidad de reutilización, actualizaciones en línea, etc. Para cada uno de éstos factores se debe establecer un nivel de influencia[9] en una escala de cero (no aplicable) a 5 (muy influyente.) Estas influencias se suman para obtener el $TDI$ (total degree of influence) y así encontrar el denominado "value added factor":

$$VAF = 0{,}65 + (0{,}01 * TDI)$$

Finalmente los puntos de función se encuentran mediante:

$$FP = UFP * VAF$$

[8]Capers Jones (2008, p.598) proporciona un listado (que señala no ser exhaustivo) conteniendo 38 variantes.

[9]En el estándar de IFPUG se señalan 14 factores. Notar que la evaluación de sus niveles de influencia a fin de cuentas está "basada en el experto".

Los puntos de función se utilizan en diversas organizaciones como mecanismo de dimensionamiento de un proyecto de software, puesto que la "funcionalidad" es un concepto relativamente accesible por los usuarios; asimismo se emplea como dato de entrada de algunas fórmulas experimentales de estimación de esfuerzo.

En Capers Jones (2008) se estudian ampliamente los puntos de funcion (centrado en el estándar IFPUG) así como información estadística del software norteamericano basada en esta métrica para diversas industrias. Una revisión muy informativa acerca de las motivaciones técnicas, ventajas y desventajas de las principales alternativas a los puntos de función, se puede encontrar en Lokan (2005): además de una recapitulación desde sus orígenes, esta fuente da cuenta de algunas propuestas adaptadas al software orientado a objetos como los "Object Oriented Function Points" (basado en diagramas de clases[10]), "Use Case Points", etc. (Lokan, 2005, p.337)

La necesidad de dimensionar el producto en desarrollo y/o el esfuerzo involucrado ha promovido el surgimiento de diversas métricas basadas o al menos inspiradas en los puntos de función para la mayoría de nichos importantes; verbigracia, "Web Object Points" e "Internet Points" como alternativa sugerida para aplicaciones web (Stutzke, 2005, p.216).

Un descendiente lejano de los puntos de función se puede encontrar en el contexto de las metodologías ágiles, donde el uso del "User Story" como bloque constituyente ha dado origen a los "Story Points" como unidad de medida de extremo pragmatismo:

> El número de 'story points' asociado a una 'story' representa el tamaño total de la 'story'. No hay una fórmula para definir el tamaño de una 'story'. Por el contrario, un estimado 'story point' es una amalgama de la cantidad de esfuerzo involucrado en desarrollar la funcionalidad, la complejidad de su desarrollo, el riesgo inherente a ésta, etcétera. (Cohn, 2005, p.36)

Desde otro ángulo, el mismo autor señala:

> Lo bello de esto es que la estimación en 'story points' separa completamente la estimación del esfuerzo de la estimación de la duración. Por supuesto, el esfuerzo y la duración están relacionados, pero el separarlos permite que cada uno sea estimado independientemente. En realidad tú ya no estás estimando la duración del proyecto; tú estás calculándola o derivándola. (Cohn, 2005, p.39)

[10]Actualmente corresponde al uso de diagramas UML (Zhou y col., 2014).

Como comentario final, señalemos que los puntos de función proporcionan una métrica que suele ser empleada como dato de entrada para otros modelos de estimación de esfuerzo. Es interesante mencionar que algunos estudios (por ejemplo Zheng y col. (2009)) utilizan los puntos de función como equivalentes (salvo una constante multiplicativa) al esfuerzo de los proyectos. Sin embargo, la mayoría de modelos proponen una relación donde el esfuerzo es proporcional a alguna potencia (que varía según el contexto) del tamaño del software (expresado en puntos de función y/o líneas de código fuente) u otras relaciones funcionales (Matson, Barrett y Mellichamp, 1994).

#### 1.2.1.5. Análisis de Regresión Lineal

La regresión lineal es el componente principal de la mayoría de modelos de estimación de esfuerzo que utilizan información histórica para sus pronósticos. Experimentalmente se ha observado que el esfuerzo de los proyectos no guarda una relación lineal con ningún métrica individual; sin embargo, existe evidencia que sugiere relaciones potenciales o exponenciales; en ese sentido es usual transformar los datos mediante funciones logarítmicas previamente a la aplicación de la regresión.

La regresión lineal (con transformación logarítmica) es discutida en el contexto de proyectos web en Mendes (2007) y Mendes y Mosley (2006) y validada experimentalmente en Čeke, Milašinović y B (2015) donde se combina con el uso de los puntos de función COSMIC (Cosmic, 2015) para la estimación del tamaño del software.

Whigham, Owen y Macdonell (2015) señalan la conveniencia de un modelo referencial (baseline model) a fin de comparar objetivamente el éxito relativo de los modelos.

Con este fin proponen un modelo basado en regresión lineal multivariable denominado "Automatically Transformed Linear Baseline Model" o ATLM que en definitiva consiste en aplicar este método estadístico sobre los datos históricos previamente transformados "convenientemente".

Estas transformaciones intentan remover el sesgo inherente a los datos provenientes de proyectos de desarrollo de software. Para decidir la bondad de las transformaciones emplean el indicador de sesgo muestral basado en momentos de la muestra:

$$b_1 = \frac{m_3}{s^3} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}^3}$$

La transformación en sí puede ser de los siguientes tipos:

1. Sin transformación (caso trivial)
2. Logarítmica
3. Raíz cuadrada

Estas transformaciones se aplican sobre cada una de las variables a la búsqueda de aquella que minimiza el indicador de sesgo. Con los datos transformados en forma óptima se procede a obtener el modelo lineal.

Finalmente se realiza el análisis de la bondad del modelo, para lo cual los autores recomiendan el uso de la métrica denotada por "$RE*$" (Relative Absolute Error) definida mediante

$$RE* = \frac{var(residuos)}{var(respuesta)}$$

donde $var(.)$ es la varianza de los datos (población), $residuos$ corresponde a las diferencias "$prediccion - medido$" y $respuesta$ corresponde a los valores resultantes de la aplicación del modelo.

En esta misma referencia los autores discuten otras medidas de bondad de ajuste (notablemente el $MMRE$, mean magnitude of the relative error) a las que recurren al momento de comparar sus hallazgos con los de otras publicaciones.

#### 1.2.1.6. COCOMO

Sin duda el modelo más desarrollado y descrito en la literatura para esta categoría corresponde al Constructive Cost Model (COCOMO.) (B. W. Boehm, 1981) En su segunda versión (conocida como COCOMO II) (B. W. Boehm, Madachy y Steece, 2000) se proporciona un modelo derivado estadísticamente a partir de información histórica proveniente de 163 proyectos de software. Aquí la estimación de esfuerzo viene dada por la fórmula:

$$PM = A * Size^E * \prod_{i=1}^{n} EM_i$$

donde $PM$ es el número de meses-persona (el esfuerzo de recursos humanos), $A$ es una constante igual a 2,94, $Size$ es el "tamaño" del software (en miles de líneas de código, KSLOC), $E$ es un exponente variable con el contexto de desarrollo relativo a la aplicabilidad de lo que comúnmente se denomina "economía de escala", y $EM_i$ son "multiplicadores de esfuerzo" que corresponden a factores de ajuste que también varían en función del contexto de desarrollo. Las líneas de código preferentemente se obtienen mediante la aplicación de "puntos de función IFPUG" y tablas de conversión que aplican un factor dependiendo del lenguaje de programación.

Como vimos, el exponente $E$ representa la capacidad del proyecto de aprovechar el concepto de economía de escala; esto es frecuente en escenarios de trabajo altamente homogéneo, donde la repetitividad de las tareas permite que éstas se completen cada vez más aprisa ya sea por el aprendizaje del ejecutante, como por las herramientas auxiliares de desarrollo que ocasionalmente se crean durante el proyecto:

$$E = 0{,}91 + 0{,}01 * \sum_{j=1}^{5} SF_j$$

donde $SF_j$ son los "factores de escala" experimentalmente tabulados; estos incluyen: similitud con productos anteriores, flexibilidad de la conformidad a especificaciones, cohesión del equipo, etc. Estos factores consiguen que $E$ se establezca cercano a la unidad, lo que tiene el efecto de modificar a $Size$ ya sea incrementándolo como reduciéndolo (casos de "economía" y "deseconomía" de escala.)

Los factores $EM_i$ son factores (relativamente cercanos a la unidad) que ajustan el esfuerzo; estos factores son 17 en COCOMO II y se obtienen de tablas obtenidas experimentalmente. A modo de ejemplo, el factor "RELY" (fiabilidad) presenta la siguiente tabla[11]:

| Calificación | Muy bajo | Bajo | Nominal | Alto | Muy alto |
|---|---|---|---|---|---|
| EM | 0.82 | 0.92 | 1.0 | 1.10 | 1.26 |

Donde la calificación asociada es:

| Descripción | Calificación |
|---|---|
| Pequeñas Inconveniencias | Muy bajo |
| Pérdidas Reducidas | Bajo |
| Pérdidas Moderadas | Nominal |
| Alta Pérdida Financiera | Alto |
| Riesgo de Vida Humana | Muy alto |

COCOMO (considerando colectivamente sus versiones I y II) es largamente el modelo más discutido en la literatura, y es reportado como el método formal más utilizado (Osmanbegović, Suljić y Agić, 2017, p.203). Es empleado en numerosos estudios como referente a ser "superado" en cuanto a su capacidad predictiva. En Barry W Boehm y Valerdi (2008), los autores reflexionan acerca los logros de COCOMO como descendiente de una larga historia de modelos de estimación que lo anteceden.

[11]Extraída de B. W. Boehm, Madachy y Steece (2000, p.41) y traducida del inglés.

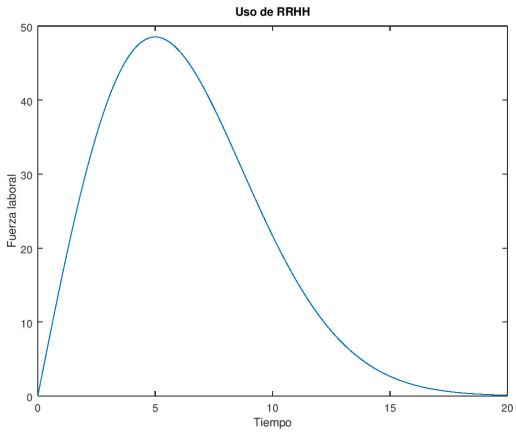



Figura 1.2.1: Curva de Norden para fuerza de trabajo.

#### 1.2.1.7. Modelo de Norden/Putnam

En base a observaciones experimentales sistemáticas, Norden (1977) observó que la asignación del personal (el esfuerzo instantáneo aplicable) a lo largo del ciclo de vida de un proyecto, puede modelarse mediante la ecuación

$$y' = 2Kate^{-at^2}$$

donde $y'$ es la asignación de personal en un determinado momento, $K$ representa el esfuerzo teórico total del proyecto, y $a$ es una constante tal que $a = \frac{1}{2t_d^2}$ donde $t_d$ es el momento en el que $y'$ se hace máximo.

Esta ecuación coincide con la función de densidad probabilística de Rayleigh descrita en las ciencias físicas[12].

La integración de esta función proporciona el esfuerzo acumulado para el proyecto:

$$y = \int_0^t y' = K(1 - e^{-at^2})$$

Esta magnitud está dada en personas-año u otra unidad análoga. Se observa claramente que:

$$\lim_{t\to\infty} y(t) = K$$

es decir, el esfuerzo total del proyecto es $K$, lo que prueba la anterior indicación que se hizo para este parámetro.

Putnam (1978) aplicó estos resultados a proyectos de software, extendiendo

[12]La distribución de Rayleigh se observa cuando la la magnitud de un vector aleatorio se relaciona con sus componentes. En particular, si $x$ e $y$ son variables aleatorias con distribución normal y media cero con la misma varianza, entonces $\sqrt{x^2 + y^2}$ sigue la distribución de Rayleigh (Papoulis y S. U. Pillai, 2002).

sus resultados y contribuyendo a su popularización. Este modelo se ha implementado en una serie de herramientas computacionales conocidas como SLIM (Software Lifecycle Management), nombre por el cual también se conoce al modelo.

**Proceso de Modelamiento** En esta sección describimos brevemente el proceso de modelamiento del modelo Putnam/Norden. Siguiendo a Putnam (1978), la velocidad de consecución de la tarea (el proyecto) es proporcional al ritmo de trabajo, así como a la cantidad de labor por ser realizada. Si $w(t)$ es la proporción de la tarea que ha sido terminada en el tiempo $t$, y $p(t)$ es la capacidad (habilidad) del personal en el tiempo $t$, entonces:

$$\overset{\circ}{w(t)} \ \alpha \ p(t)$$

$$\overset{\circ}{w(t)} \ \alpha \ (1 - w(t))$$

Esto conduce a la ecuación diferencial:

$$\frac{dw(t)}{dt} = p(t)(1 - w(t)) \tag{1.2.1}$$

que corresponde a un problema de Cauchy pues $w(0) = 0$ (en $t = 0$ no se ha completado nada de la tarea.) Resolviendo:

$$\frac{dw}{1-w} = p(t)dt \Rightarrow -ln(1-w) = \int p \Rightarrow 1 - w = e^{-\int p} \Rightarrow$$

$$w(t) = 1 - exp(-\int_0^t p(s)ds)$$

Norden asumió una "curva de aprendizaje lineal":

$$p(t) = at \Rightarrow w(t) = 1 - exp(-\frac{at^2}{2}) \Rightarrow$$

$$\overset{\circ}{w(t)} = at.exp(-\frac{at^2}{2})$$

lo que corresponde a la ecuación de la curva de Norden/Rayleigh.

**La "Ecuación del Software"** A partir de la ecuación de Norden, Putnam introdujo algunas asunciones complementarias y dedujo lo que denominó la "ecuación del software":

$$S_s = C_k K^{1/3} t_d^{4/3}$$

Aquí $S_s$ el el "tamaño" del software desarrollado en el proyecto, y $C_k$ es una constante que representa el "estado de la tecnología" del momento.

En teoría, esta ecuación permite efectuar una planificación óptima de los recursos para un proyecto; sin embargo, en la práctica es poco utilizada dadas algunas consecuencias paradójicas de su aplicación, así como por la falta de rigor en su deducción y verificación tal como se detalla en diversas referencias [13].

**Variantes al modelo de Putnam** El modelo de Putnam tuvo un considerable impacto académico y consecuentemente fue criticado por diversos investigadores, algunos de los cuales propusieron sendas alternativas y/o extensiones a su modelamiento. Verhoef (2002, p.53) proporciona una recapitulación crítica de algunas propuestas en esta línea, las cuales suelen partir de una generalización del modelo de Norden:

$$2Kate^{-at^2} \Longrightarrow at^{\alpha}e^{-bt^{\beta}}$$

#### 1.2.1.8. Leyes de la Evolución del Software

Otro antecedente más lejano pero ampliamente mencionado en la literatura corresponde a las denominadas "leyes de evolución del software" de Lehman (M. M. Lehman, 1980), quien propone una serie de postulados de tipo cualitativo y de aplicación muy general para los aplicativos de software. A partir de estas ideas se han producido diversos modelos cuantitativos que describen aspectos de la evolución del "ciclo de vida" del aplicativo (Belady y Meir M Lehman, 1976; Riordan, 1977; Woodside, 1979) (especialmente alrededor del tamaño y la complejidad) en relación a diversas fuerzas externas y contrapuestas del entorno que ocasionan sus cambios; esto es, la presencia de lazos de realimentación, así como de métricas con comportamiento cíclico; esto último típicamente relacionado a las sucesivas "versiones" o "lanzamientos" (releases) del producto.

Estas leyes de evolución del software, pese a considerar el "ciclo de vida" en su completitud, ponen énfasis en los eventos asociados a aplicativos con un largo tiempo de uso, en detrimento de los que aún están siendo confeccionados en un proyecto de desarrollo; es decir, su aplicabilidad se orienta hacia lo que se denomina "mantenimiento" del software:

> Hoy en día, la evolución del software se ha constituido en un campo de investigación muy activo y respetable, y los términos evolución de software y mantenimiento de software son con frecuencia usados como sinónimos. (Mens y Demeyer, 2008).

[13] Consultar por ejemplo Suelmann (2014) y B. Kitchenham, Taylor y R (1984).

### 1.2.2. Acerca de los Modelos de Estimación de Esfuerzo

En estas secciones discutimos algunos aspectos comunes a los estudios relacionados a modelos de estimación de esfuerzo; en especial, señalamos una serie de problemas recurrentes y que a nuestro criterio invalida gran parte (sino la gran mayoría) de las propuestas descritas en la literatura.

**a) Desarrollo a lo grande y desarrollo en pequeño** Al consultar los orígenes de los primeros modelos de estimación de esfuerzo se aprecia rápidamente que sus principales proponentes han pertenecido a grandes organizaciones embarcadas en proyectos de elevada complejidad a cargo de grandes equipos de especialistas. Esto se refleja muy claramente en las curvas de Norden antes reseñadas, las cuales sólo tienen sentido en proyectos con estas características.

Esta visión del desarrollo de software "a lo grande" (donde el computador todavía es un privilegio muy exclusivo), corresponde perfectamente a las décadas 50-70, donde los proyectos en cuestión son principalmente de índole científico/-militar y/o gubernamental[14] . Es una era de especialistas atendiendo a otros especialistas, empleando un conjunto aún muy reducido de tecnologías.

Con la llegada del microcomputador (décadas 80-90) podemos encontrar especialistas atendiendo a un numeroso grupo de no-especialistas. El software automatiza diversos procesos de organizaciones de todo tamaño y especie, y se convierte en herramienta de trabajo para toda clase de profesionales y entusiastas.

Este proceso de masificación del software impulsa el desarrollo de múltiples emprendimientos a pequeña escala, para lo cual se conforman equipos de desarrollo dedicados a nichos cada vez más restringidos a fin de satisfacer las necesidades más especializadas de los clientes. En consecuencia, estos equipos son ahora de tamaño muy reducido y deben ejecutar los desarrollos en plazos cada vez más ajustados: sus integrantes son normalmente los mismos desde el principio hasta el final del proyecto puesto que no hay tiempo para reclutamientos sobre la marcha; asimismo, es bien sabido que el personal que se incorpora a un proyecto ya en curso, normalmente lo retrasa (Brooks Jr, 1995). A esta forma de enfrentar los proyectos le denominamos desarrollo "en pequeño".

Esto quiere decir que la dinámica de los proyectos en la actualidad es radicalmente distinta a la que se daba en los albores de la ingeniería de software; naturalmente, también existen proyectos de gran envergadura en ciertos nichos; sin embargo, en términos relativos el desarrollo de proyectos "en pequeño" representa largamente la mayor proporción en esta actividad: en este nuevo escenario

[14]Poco tiempo después se incluye el soporte a la gestión comercial de grandes empresas y la banca. Una detallada recopilación histórica para la ingeniería de software subdividida en décadas puede consultarse en Capers Jones (2013).

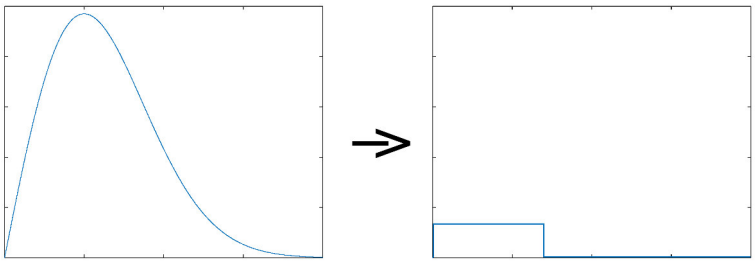


Figura 1.2.2: Proyectos “a lo grande” vs. desarrollo “en pequeño”.

de pequeños equipos (de tamaño esencialmente invariante) operando con plazos muy breves, la antigua curva de Norden se convierte en una función constante de reducida magnitud y duración. Esta idea se ilustra en la figura 1.2.2.

Lo anterior implica que los modelos de estimación tradicionales que describen la variación temporal de un personal numeroso, son sencillamente inaplicables para los proyectos que usualmente acometen los desarrolladores de hoy.

**b) Un mundo ágil: la variabilidad de los requerimientos** El esfuerzo requerido en un proyecto depende directamente de las necesidades (requerimientos) a satisfacer; lógicamente, un conjunto distinto de requerimientos involucrará un esfuerzo distinto. Sin embargo, es el caso que los requerimientos de cualquier proyecto de software no trivial de nuestros días *siempre* resultan ser distintos a los que se consideraron inicialmente, por lo que el estimado inicial *siempre* se torna (en diverso grado) inútil.

Luego, una fuerte variación en los requerimientos tiene como correlato una fuerte variación en el esfuerzo involucrado en el desarrollo. En ese sentido, es menester indagar acerca del grado de variabilidad de los requerimientos en los proyectos: si nos atenemos a las tendencias más populares en el desarrollo de software, debemos concluir que esta variabilidad sólo ha ido en aumento a través de los años.

En el plano metodológico, con el cambio de milenio se han popularizado ampliamente las denominadas “metodologías ágiles” donde el énfasis gira en torno a la plena satisfacción del cliente, para lo cual sus requerimientos son re-elaborados iterativamente puesto que su conocimiento a priori se asume imposible. Luego, la continua variación en los requerimientos se acepta como inexorable e incluso benéfica. Como ilustración, uno de los principios del ya clásico “Manifiesto ágil” reza:

“Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.”(Beedle, 2001)

En el frente tecnológico, la ubicuidad del Internet ha elevado la representatividad numérica de los desarrollos de software orientados a la web. Al respecto Glass (2004) reflexiona irónicamente acerca de los proyectos web:

- Libres de requerimientos (los clientes aún no saben lo que quieren realmente)
- Desarrollo rápido (los clientes pueden no saber lo que quieren, ¡pero saben que lo quieren pronto!)
- Poca duración y pequeñez (hablan de desarrollo "3x3" - tres personas por tres meses.) (p.102.)

En la misma línea Mendes (2014) señala:

¿Cuál de las técnicas proporciona la predicción más precisa para la estimación del esfuerzo Web? A la fecha, la respuesta a esta pregunta ha sido simplemente: "depende". [...] no se ha alcanzado consenso en cuanto a qué técnica de esfuerzo tiene el mejor poder predictivo, no obstante haberse ejecutado estudios comparativos por casi 20 años (p.47.)

La conclusión lógica apunta a que el problema de estimación en los últimos años se ha tornado cada vez más difícil, especialmente en entornos con requerimientos de alta variabilidad.

**c) Los "Datos Históricos" que justifican los Modelos** Para la sustentación de los modelos de estimación de esfuerzo, muchos investigadores han recurrido a bases de datos históricas públicas ("datasets" en adelante), que consolidan diversas métricas de proyectos ya ejecutados, pero cuya idoneidad deja mucho que desear por diversos motivos.

El primer inconveniente corresponde a la ausencia de información necesaria: los datos deben provenir de proyectos reales y contener precisamente la información requerida por los modelos a ser contrastados; evidentemente, esto no siempre es posible, pues las costumbres de recolección de métricas de proyectos varían de organización en organización. Esto significa que diversos modelos nunca alcanzan a ser puestos en práctica con información estadísticamente significativa, por la simple razón que los datos necesarios no están disponibles. Según Capers Jones (2008) la mayoría de empresas (de desarrollo de software) simplemente no registra datos, salvo un pequeño subconjunto de las actividades realizadas (p.193.)

En segundo lugar, la calidad de los datos es cuestionable. En las últimas dos décadas se han acumulado diversas bases de datos de proyectos, notablemente las que actualmente están organizadas en el ISBSG (ISBSG, n.d.), así como las documentadas en C. Jones (2011). Los datos recolectados en estos repositorios son producto de la "donación" de diversas organizaciones, las cuales no tienen mayor

incentivo para asegurar su veracidad y/o exactitud. Capers Jones (2008) señala como un hecho lamentable que la mayoría de sistemas de seguimiento corporativo para los costos y el esfuerzo (dólares, horas de trabajo, meses-persona, etc.) son incorrectos y logran omitir desde 30 a más de 70 por ciento del esfuerzo real aplicado a los proyectos de software. (p.192)

En esta línea Hetzel (1995) da cuenta de la inutilidad de la gran mayoría de mediciones y estimaciones publicadas por la industria del software: se recolecta grandes cantidades de información acerca de prácticas que están pobremente descritas o son defectuosas desde el principio. Esa información es entonces diseminada en forma tal que hace casi imposible de confirmar o validar su significancia. (p.172)

De otro lado, Capers Jones (2017) nos advierte que los datos publicados están representados en unidades incompatibles: la industria del software es única en tener más variantes métricas que cualquier otra disciplina de ingeniería en la historia, combinada con una casi total ausencia de reglas de conversión de una métrica a otra. (p.221)

Siguiendo a este mismo autor, la dimensión temporal tampoco está exenta de crítica: La industria de software al año 2016 no tiene ningún método estándar para medir ya sea la fecha de inicio o de término de los proyectos de software mayores. Como resultado, la información histórica de su "programación cronológica" es altamente sospechosa. (p.158)

Otro efecto negativo se advierte cuando se tiene en consideración que un porcentaje considerable de los proyectos son cancelados antes de su conclusión[15], lo que ocurre usualmente cuando el esfuerzo dedicado (y/o el tiempo transcurrido) sobrepasa algún umbral de tipo comercial o contractual. En tal escenario el proyecto usualmente no es reportado, y su ausencia implica que los modelos poseen un fuerte sesgo hacia aquellos proyectos que (por cualesquiera razones) sí lograron ser exitosos.

Somos de la opinión que esta situación se deriva especialmente de la dificultad y/o reticencia que tienen los administradores y sus organizaciones para admitir sus problemas de gestión de proyectos. Estos problemas rara vez son generados por incompetencia o falta de interés, sino por el contexto de alta incertidumbre usualmente asociado a los proyectos de software: requerimientos vagamente definidos y prioridades cambiantes; interrupciones para atender emergencias operati-

[15]Existen diversos estudios (normalmente basados en encuestas) acerca de la tasa de fallo de los proyectos. Por ejemplo, el estudio de (Emam, Koru y G, 2008) realizado en 2005 y 2007 intenta superar algunas contradicciones y ausencias frecuentes en las publicaciones acerca de este tópico. En sus conclusiones señala: La tasa de cancelación de proyectos de TI estuvo en el rango de 11.5 a 15.5 %. [...] La tasa combinada de proyectos cancelados pero no exitosos estuvo entre 26 % y 34 %.

vas, rotación del personal con conocimiento privilegiado, sobretiempos de dudosa legalidad, tiempo "robado" para la investigación de las nuevas tecnologías, etc. Todos estos factores son inevitables (si bien pueden ser mitigables), pero tienen en común que exponen a los especialistas a la crítica, así como a demandas legales contra sus organizaciones.

En conclusión, los datos históricos que suelen emplearse para la justificación empírica de los modelos son incompletos, imprecisos, incompatibles y sesgados. Sin embargo, esto no ha sido obstáculo para que sigan siendo empleados en la calibración de los modelos tradicionales y experimentales, así como en sendas contrastaciones acerca de su potencial efectividad.

**d) La precisión de los modelos en relación al tiempo** En todas las disciplinas que poseen alguna clase de modelo predictivo, el nivel de precisión numérica asociado a sus resultados (o predicciones) constituye un elemento imprescindible para su aplicación efectiva. Sin embargo, en los modelos de estimación de esfuerzo de software esto se ignora o se considera muy a la ligera. Por ejemplo, McConnell (2006) recomienda calcular los rangos de incertidumbre de los estimados a partir del valor puntual obtenido por el modelo, aplicando un conjunto de factores que son extraídos de la literatura de COCOMO (p.39,253); estos rangos son dinámicos, en el sentido que la estimación se hace cada vez más precisa conforme discurre la ejecución del proyecto[16].

Sin embargo, el problema radica en el pobre nivel de precisión de tales predicciones consideradas "exitosas". Concretamente, para una estimación de esfuerzo "$X$" obtenida al inicio del proyecto (cuando tal estimación tiene mayor valor), los autores mencionados reportan una (muy elevada) incertidumbre correspondiente al rango $[25\,\%X - 400\,\%X]$. Este intervalo se reduciría paulatinamente conforme se avanza con el proceso de análisis, concluyendo en un excelente $[90\,\%X - 110\,\%X]$ al completarse las tareas denominadas de "diseño detallado".

Lamentablemente, en la práctica es menester estimar y presupuestar los proyectos antes que éstos sean iniciados[17], justamente cuando los modelos proporcionan respuestas con excesivo margen de error y -por tanto- de poca valía.

Este problema se exacerba para los proyectos ejecutados con metodologías no tradicionales en las que el diseño evoluciona en paralelo con la construcción

[16]El gráfico típico de esta idea se conoce como el "cono de incertidumbre"; éste puede consultarse en las referencias mencionadas en esta sección. Por su parte, la referencia original de COCOMO (B. W. Boehm, 1981) al describir tales factores sólo menciona (en un pie de página) que tales rangos han sido determinados *subjetivamente*, y representan límites de confianza al 80 % (p.310). Siguiendo la práctica estadística usual, esto se debe interpretar como modelos cuyo "éxito" predictivo ocurre en aproximadamente el 80 % de ocasiones.

[17]Usualmente dependerá del presupuesto si el proyecto se inicia, se posterga, se reformula, o se cancela.

iterativa del código fuente: la estimación de esfuerzo (y el presupuesto asociado) también estaría sujeta a evolucionar.

**e) La Objetividad de los Datos de Entrada** Diversos modelos de estimación utilizan como punto de partida (o dato de entrada principal) al "tamaño total" del código fuente de los programas a ser creados en el proyecto en cuestión: este tamaño es considerado el principal determinante para el esfuerzo asociado.

El tamaño del código fuente se debe obtiener a partir de los requerimientos del proyecto, usualmente mediante el análisis de puntos de función u otros métodos análogos, en donde normalmente interviene la opinión del experto.

Si estos modelos de estimación en realidad no parten de los requerimientos del proyecto, sino de un "tamaño estimado" (más o menos subjetivo) proporcionado por los especialistas, entonces sus resultados necesariamente acarrean esta carga subjetiva: su exactitud estará siempre supeditada a la calidad de la opinión del experto. Lamentablemente, un modelo con resultados de exactitud desconocida, es esencialmente inútil.

Este problema no sólo se manifiesta con el estimado del tamaño total, sino con la medición y/o calificación de los datos de entrada complementarios; para ilustrar este punto, veamos un ejemplo tomado de COCOMO II (B. W. Boehm, Madachy y Steece, 2000). En este modelo la "cohesión del equipo" es un dato de entrada (conocido como "factor") que se define así:

El factor escala de cohesión del equipo da cuenta de las fuentes de turbulencia y entropía del proyecto debido a dificultades en sincronizar a los actores relevantes del proyecto. (p.34)

El modelo prescribe el uso de una tabla de escala Likert (de cinco opciones) que debe ser empleada por el usuario del modelo para "calificar" al factor. Por tanto, el especialista deberá ser capaz de predecir si "el equipo" (que podría incluir personal recién reclutado) será -a lo largo del proyecto- muy cohesionado, algo cohesionado, neutralmente cohesionado, poco cohesionado, o muy poco cohesionado.

Esta forma de calificación o "medición" resulta ser de naturaleza totalmente subjetiva. En la práctica, esta tarea será encargada a un experto responsable del proyecto, con lo cual -tras un gran rodeo- se termina reintroduciendo (por la puerta falsa) la opinión subjetiva de los especialistas.

Todo lo anterior nos conduce a proponer una obvia recomendación de cara a nuevos modelos de estimación: en lo posible, éstos deberían ser alimentados *directamente* por los requerimientos del proyecto. Sin duda esto no es sencillo, puesto que los requerimientos usualmente son especificados de un modo extremadamente informal, y con mucha frecuencia provienen de actores no especializados.

En este orden de ideas, resulta razonable evaluar la introducción de una espe-

cificación intermedia formalizada para los requerimientos[18] como etapa previa a la aplicación de los modelos, a fin de proporcionarles datos de entrada objetivos. Notar que el uso de especificaciones formalizadas incrementaría las oportunidades de automatización en las futuras herramientas de estimación.

**f) La Contrastación de los Modelos** Los investigadores interesados en la estimación del esfuerzo del software han dedicado mucho esfuerzo a la comparación empírica de los modelos a fin de dilucidar cuáles son los más convenientes y/o recomendables. Estas comparaciones requieren (al menos) de un amplio conjunto de datos históricos de alta calidad, así como de algunas métricas objetivas de contrastación a ser aplicadas sobre los resultados estimados. Como ya vimos anteriormente, la industria no ha logrado generar los datos históricos necesarios; sin embargo, las métricas utilizadas tampoco han escapado a la crítica.

Siguiendo a Jørgensen (2007), el problema fundamental con estas métricas de comparación de modelos se deriva de la inexistencia de un acuerdo en lo que significa que un estimado sea mejor que otro. Concretamente, la métrica más utilizada en la literatura para comparar la capacidad predictiva de los modelos es la "media del error total relativo" (MMRE) [19], y como en toda métrica basada en el error, un valor más reducido se interpreta como una mayor exactitud (alcazada por el modelo); sin embargo, en la práctica esto resulta ser falso en muchos escenarios[20]. En esta línea, Barbara A Kitchenham y col. (2001, p.81) proporciona un análisis acerca de las propiedades estadísticas del MMRE, señalando que esta métrica no es efectiva para evaluar la exactitud de las predicciones; por el con-

[18]Existen diversos lenguajes formales de modelamiento de requerimientos (Jureta, 2015, p.33), los cuales son un punto de partida natural para lo que proponemos. Estos lenguajes capturan totalmente el problema que intenta resolver el proyecto, permitiendo la aplicación de métodos formales orientados a garantizar la corrección al momento de la ejecución de los programas. Lamentablemente su uso conlleva una elevada complejidad y considerable esfuerzo, lo que es prohibitivo en nuestro escenario. En ese sentido, futuras investigaciones deben explorar alternativas simplificadas y orientadas exclusivamente a la estimación.

[19]Denotando por $x_i$ al esfuerzo invertido en un proyecto, si la estimación por algún modelo modelo fue $\widehat{x_i}$, entonces se define la magnitud del error relativo (Magnitude of Relative Error, MRE) como:

$$MRE = |\frac{\widehat{x_i} - x_i}{x_i}|$$

A su turno, el MMRE (Mean Magnitude of Relative Error o Media del error total relativo) para un conjunto de estimaciones (aplicación del modelo a múltiples proyectos) se define mediante:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} |\frac{\widehat{x_i} - x_i}{x_i}|$$

Para otras métricas relacionadas, consultar por ejemplo Shepperd, Cartwright y Kadoda (2000) y Sharma y col. (2010).

[20]Jørgensen (2007) ilustra este punto con un ejemplo: un estimado de esfuerzo para el cual el esfuerzo real es 300 % del resultado de la estimación (MRE=0.67) resulta ser *más exacto* que un estimado de esfuerzo donde el esfuerzo real es 59 % del estimado (MRE=0.69.) (p.4)

trario, correspondería a una métrica de dispersión de la variable aleatoria dada por $\frac{\widehat{x_i}}{x_i}$.

Estas métricas de dudosa fundamentación al ser aplicadas sobre datos aún más cuestionables, explican una larga cantidad de estudios no concluyentes y/o mutuamente contradictorios acerca de las supuestas bondades predictivas de ciertos modelos en relación a otros, lo que sin duda menoscaba la reputación de este campo de estudio en su conjunto.

### 1.2.3. Modelos de Fiabilidad

Los modelos de fiabilidad de software se inician desde principios de la década de los setenta; estos conceptos se adaptan de la teoría de fiabilidad del hardware:

> El enfoque básico es modelar datos de fallos del pasado para predecir el comportamiento futuro. Este enfoque emplea tanto el número de fallos descubiertos por periodo de tiempo como el tiempo observado (de reloj real o alguna medida de tiempo de ejecución de computador) entre fallos del software. Lyu y col. (1996, p.72)

Tradicionalmente, estas investigaciones están comprendidas en las subdisciplinas de control de calidad y/o aseguramiento de la calidad. Para el presente estudio consideramos relevantes a los denominados "modelos de crecimiento de fiabilidad de software" (software reliability growth models, SRGM.) En Huang, Kuo y Lyu (2007) se puede consultar un estudio comparativo de diversas investigaciones de esta clase.

En muchos proyectos de software formalmente administrados es usual que el desarrollo (del aplicativo o de algún componente) se dé por concluido cuando sus defectos se reducen por debajo de cierto umbral. Estos modelos proporcionan estimaciones objetivas acerca de los defectos presentes en el software (incluso aquellos no descubiertos) a lo largo del tiempo, por lo que en realidad corresponden a una "dinámica" del software desde la perspectiva de la calidad del producto (Staron y col., 2013).

El modelo de Goel y Okumoto (Amrit L Goel y Okumoto, 1979) es una de las propuestas iniciales y de mayor influencia en relación a la dinámica de fiabilidad de software, por lo que se introduce en esta sección como antecedente ilustrativo. Consultar el capítulo 5 de Pham (2007) para una revisión de diversos modelos de fiabilidad de software.

#### 1.2.3.1. El modelo de Goel-Okumoto

Este modelo (Amrit L Goel y Okumoto, 1979) se basa en la aplicación del proceso de Poisson no homogéneo con las siguientes asunciones:

1. El número de fallos acumulados $M(t)$ sigue un proceso de Poisson no homogéneo con media $m(t)$.

2. El número de fallos esperados en un intervalo $[t, t + \Delta t]$ es proporcional al valor esperado del número de fallos no detectados en el tiempo $t$.

3. La media es una función acotada y no decreciente; es decir, $\lim_{t\to\infty} m(t) = a < \infty$ (o también $\lim_{t\to\infty} \lambda(t) = 0$.)

Asimismo, al inicio no se ha descubierto fallo alguno, por lo que se tiene la condición inicial: $m(0) = 0$.

El número de defectos no detectados en el tiempo $t$ es $(a - M(t))$ con valor esperado $(a - m(t))$, luego por la segunda asunción:

$$m(t + \Delta t) - m(t) = b(a - m(t))\Delta t$$

Haciendo $\Delta t \to 0$ se obtiene la forma diferencial:

$$m' = b(a - m)$$

Al integrar esta EDO considerando la condición inicial se obtiene

$$m(t) = a(1 - e^{-bt}) \tag{1.2.2}$$

Es fácil ver que esta función satisface las condiciones inicial y final, presentando una caída suave conforme $t \to \infty$ lo que se interpreta como una fuerte reducción de los defectos tras largo tiempo de operación. Del mismo modo la función de intensidad es:

$$\lambda(t) = abe^{-bt} \tag{1.2.3}$$

esta función es estrictamente decreciente y tiende a cero en el infinito. Se interpreta como una gradual escasez de descubrimiento de defectos: cuando ha transcurrido suficiente tiempo, la ocurrencia de un fallo será un suceso muy poco probable. Entre los practicantes del software esto se identifica con la madurez que alcanza un aplicativo cuando éste ha acumulado mucho uso, y ha recibido las correcciones necesarias.

En la misma referencia (Amrit L Goel y Okumoto, 1979), los autores también proporcionan una función de máxima verosimilitud (maximum likelihood) (Pham, 2007, p.79) a fin de ajustar los parámetros a partir datos tabulares.

Una generalización de los autores (Amrit L. Goel, 1985) corresponde a lo que denominan el Modelo de Poisson no Homogéneo Generalizado de Goel, concebido

para reflejar algunos patrones observados durante las pruebas del software. En este caso la media de Poisson tiene la forma:

$$m(t) = a(1 - e^{-bt^c})$$

y la intensidad

$$\lambda(t) = abc.e^{-bt^c}t^{c-1}$$

#### 1.2.3.2. Modelamiento estocástico de Cambios en Código Fuente

Aman y Ohkochi (2010) equiparan la detección de los fallos con los cambios en el código fuente, lo que consideramos razonable si se tiene en consideración que los fallos detectados implican una pronta corrección por parte del desarrollador. Con otro enfoque, todo requerimiento de programación puede considerase trivialmente como un fallo, en el sentido que se trata de una funcionalidad que no está en operación (por no haberse implementado todavía.) Con estas ideas proponen la hipótesis complementaria:

> Ahora proponemos modelar la producción de código fuente para desarrollo de código abierto, igual que como la detección de fallos anterior: un cambio en código es considerado un evento estocástico, y el monto generado de código fuente es explicado por una curva de modelo de crecimiento basada en el Proceso de Poisson No Homogéneo. (Aman y Ohkochi, 2010, p.49)

Los autores emplean funciones sigmoidales para el conteo de errores (lo que es usual en este campo de investigación.) Así, establecen un modelamiento cuantitativo para los cambios de código fuente a través del tiempo, lo que constituye el antecedente más cercano a nuestra investigación. Como veremos, el uso del proceso de Poisson no homogéneo (NHPP) ha sido muy empleado en el análisis de fiabilidad de software:

NHPP es un modelo realista para predecir la fiabilidad de software y tiene una interpretación muy interesante y útil en el testeo y depuración del software. (Pham, 2007, p.180)

# Capítulo 2

# Bases Teóricas

En esta sección desarrollaremos los conceptos matemáticos y estadísticos que dan sustento a la presente investigación.

## 2.1. Modelo de Regresión Lineal

Esta sección se basa fundamentalmente en el apéndice C de Gujarati y Porter (2010).

### 2.1.1. Convención Matricial de Derivación Parcial

Las siguientes convenciones tomadas de Gujarati y Porter (2010) nos permitirán expresar las relaciones de regresión lineal de un modo muy compacto.

Sean las matrices de $n * 1$ elementos reales representadas mediante:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ \\ x_n \end{bmatrix}$$

y una función escalar $r: \ R^n \to R$ definimos la derivada de esta función con respecto a sus componentes mediante:

$$\frac{\partial r(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial r}{\partial x_1} \\ \frac{\partial r}{\partial x_2} \\ ... \\ ... \\ \frac{\partial r}{\partial x_n} \end{bmatrix}$$

Sea A una matriz simétrica (constante) $\boldsymbol{A} \in \mathbb{R}^{n*n}$, entonces se verifica:

$$\frac{\partial(\boldsymbol{x'}A\boldsymbol{x})}{\partial \boldsymbol{x}} = 2A\boldsymbol{x}$$

Sea el vector constante $\boldsymbol{v} \in \mathbb{R}^{n*1}$, entonces se verifica:

$$\frac{\partial(\boldsymbol{x'}\boldsymbol{v})}{\partial \boldsymbol{x}} = \boldsymbol{v}$$

En forma más clásica, estas relaciones se escriben mediante el operador nabla:

$$\nabla(\boldsymbol{x'}A\boldsymbol{x}) = 2A\boldsymbol{x} \tag{2.1.1}$$

$$\nabla(\boldsymbol{x'}\boldsymbol{v}) = \boldsymbol{v} = \nabla(\boldsymbol{v'}\boldsymbol{x}) \tag{2.1.2}$$

### 2.1.2. Sistema de Ecuaciones Lineales

Dado un conjunto de "n" muestras con valores $y_1 ... y_n$ cada una de estas se aproxima mediante una combinación lineal de "k" variables:

$$y_j \approx \beta_1 x_{1j} + \beta_2 x_{2j} + \beta_3 x_{3j} + ... + \beta_k x_{kj}$$

Es conveniente asumir que una de estas variables es la unidad, lo que nos permite ajustar un "intercepto" distinto de cero; así, $x_{1j} = 1$:

$$y_j \approx \beta_1 + \beta_2 x_{2j} + \beta_3 x_{3j} + ... + \beta_k x_{kj}$$

Esta aproximación se modela como una ecuación que contiene un término de error o perturbación $\mu_j$:

$$y_j = \beta_1 + \beta_2 x_{2j} + \beta_3 x_{3j} + ... + \beta_k x_{kj} + \mu_j$$

Matricialmente:

$$\begin{bmatrix} y_1 \\ y_2 \\ ... \\ \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{21} & x_{31} & ... & x_{k1} \\ 1 & & & & \\ ... & & & & \\ & & & & \\ 1 & x_{2n} & x_{3n} & ... & x_{kn} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ ... \\ \\ \beta_k \end{bmatrix} + \begin{bmatrix} \mu_1 \\ \mu_2 \\ ... \\ \\ \mu_n \end{bmatrix}$$

Es decir:

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{\beta} + \boldsymbol{\mu}$$

Observar que las dimensiones de $\boldsymbol{y},\ \boldsymbol{x},\ \boldsymbol{\beta},\ \boldsymbol{\mu}$ son $n*1,\ n*k,\ k*1,\ n*1$

respectivamente.

### 2.1.3. Minimización Cuadrática

Este procedimiento también se conoce como OLS: Ordinary Least Squares. Se trata de hallar un conjunto de valores $\{\beta_j\}$ tal que se minimiza el error total en el sistema; más precisamente, se minimiza $\mu_1^2 + ... + \mu_n^2 = \boldsymbol{\mu'\mu}$. De la anterior ecuación:

$$\boldsymbol{\mu = y - x\beta} \rightarrow \boldsymbol{\mu' = y' - \beta' x'} \rightarrow$$

$$\boldsymbol{\mu'\mu = (y' - \beta' x')(y - x\beta) = y'y - y'x\beta - \beta' x' y + \beta' x' x\beta}$$

Notar que todos los términos son escalares, luego $\boldsymbol{y'x\beta = (y'x\beta)' = \beta' x' y}$:

$$\boldsymbol{\mu'\mu = y'y - 2\beta' x' y + \beta' x' x\beta}$$

Esta expresión (escalar) para ser minimizada requiere que todas sus derivadas parciales con respecto a las $\beta_j$ se anulen; es decir:

$$\frac{\partial(\boldsymbol{\mu'\mu})}{\partial\boldsymbol{\beta}} = \mathbf{0}$$

Aplicando las reglas de derivación (notar que $\boldsymbol{x'x}$ corresponde a la matriz simétrica A de la primera regla), obtenemos:

$$\frac{\partial(\boldsymbol{\mu'\mu})}{\partial\boldsymbol{\beta}} = -2\boldsymbol{x'y} + 2\boldsymbol{x'x\beta} = 0 \rightarrow \boldsymbol{x'x\beta = x'y} \rightarrow$$

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{x'x})^{-1}\boldsymbol{x'y}$$

Donde $\widehat{\boldsymbol{\beta}}$ es un vector de estimadores aproximado mediante el método OLS, presumiblemente cercano al real $\boldsymbol{\beta}$ que subyace al fenómeno modelado.

### 2.1.4. Varianza y Covarianza

Para una variable aleatoria "x" que (con igual probabilidad) toma los valores $x_1, x_2, ...x_n$, su valor esperado es:

$$E(x) = \frac{\sum_{i=1}^{n} x_i}{n} = \overline{x}$$

Las "desviaciones" de cada elemento respecto al valor esperado corresponden a $(x_i - \overline{x})$ (éstas pueden ser positivas o negativas.) El valor esperado del cuadrado de estas desviaciones es la varianza de estos valores:

$$\sigma^2 = E[(x - \overline{x})^2] = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}$$

Ocasionalmente es útil la siguiente relación alternativa que se obtiene fácilmente desarrollando el binomio cuadrático:

$$\sigma^2 = \overline{x^2} - \overline{x}^2 = E(x^2) - E(x)^2$$

La desviación estándar de los mismos valores (usualmente denominada $\sigma$) es la raíz cuadrada del valor obtenido para $\sigma^2$:

$$\sigma = \sqrt{\sigma^2}$$

Los valores $x_i$ pueden provenir la medición de cierto fenómeno, en cuyo caso se trata de una muestra representativa del mismo.

De otro lado, consideremos otra variable aleatoria "y" que (con igual probabilidad) toma valores $y_1, y_2, ...y_m$. Consideremos la generación de un par $(x_i, y_j)$ constituido por valores de ambas variables aleatorias. La probabilidad de obtener un determinado $x_i$ es $\frac{1}{n}$ mientras que la de obtener a $y_j$ es $\frac{1}{m}$. Si ambas variables son independientes, entonces la probabilidad de obtener $x_i$ e $y_j$ es el producto de las anteriores: $\frac{1}{nm}$.

Considerando ahora la variable aleatoria $xy$, sus $n*m$ posibles valores tendrán todos la misma probabilidad $\frac{1}{nm}$, por lo que su valor esperado será:

$$E[xy] = \frac{\sum_{i=1:n,j=1:m} x_i y_j}{nm} = \frac{\sum_{i=1:n}\sum_{j=1:m} x_i y_j}{nm}$$

Sin embargo, al multiplicar los valores esperados de las variables se obtiene el mismo resultado:

$$E(x)E(y) = \frac{\sum_{i=1}^{n} x_i}{n}\frac{\sum_{j=1}^{m} y_j}{m} = \frac{\sum_{i=1:n}\sum_{j=1:m} x_i y_j}{nm}$$

Luego, para variables independientes: $E[xy] = E(x)E(y) = \overline{xy}$.

La covarianza entre dos variables aleatorias se define como el valor esperado del producto de las "desviaciones" de cada variable aleatoria respecto a su valor esperado:

$$Cov(x, y) = E[(x - \overline{x})(y - \overline{y})]$$

$$= E[xy - \overline{x}y - x\overline{y} + \overline{xy}] = E[xy] - E[\overline{x}y] - E[x\overline{y}] + E[\overline{xy}]$$

Considerando que si a es constante: $E[ax] = aE[x]$:

$$= E[xy] - \overline{x}E[y] - \overline{y}E[x] + \overline{xy} = E[xy] - \overline{xy} - \overline{yx} + \overline{xy} \rightarrow$$

$$Cov(x, y) = E[xy] - \overline{xy}$$

Según vimos, si las variables son independientes $E[xy] = \overline{xy}$, por lo que $Cov(x, y) = 0$.

### 2.1.5. Asunciones del Modelo de Regresión Clásico

a) Se asume que las variables aleatorias $\mu_i$ se dispersan alrededor de cero, siendo este último su valor esperado, $E(\mu_i) = 0$; en forma matricial:

$$E(\boldsymbol{\mu}) = \mathbf{0}$$

b) La varianza de las variables aleatorias $\mu_i$ se mantiene constante para todas las muestras (homoscedasticidad); es decir, la "dispersión" de los valores se no varía en la región de muestreo. A esta varianza se le denominará $\sigma^2$:

$$E(\mu_i^2) = \sigma^2$$

c) No hay correlación entre las variables aleatorias $\mu_i$, es decir:

$$E(\mu_i \mu_j) = 0 \ ; \ i \neq j$$

Tanto "b" como "c" se pueden expresar de modo compacto mediante la "matriz de varianza-covarianza" de las perturbaciones:

$$E(\boldsymbol{\mu}\boldsymbol{\mu'}) = \sigma^2 I$$

d) Colinealidad: si en cada muestra se tiene que todos los valores de $x_1$ son el doble del correspondiente a $x_2$ entonces es evidente que el modelo presenta redundancia, por lo que una de estas variables deberá ser suprimida. En general, este fenómeno corresponde a la dependencia lineal entre los vectores (columna) de $\boldsymbol{x}$. Recordando que esta matriz tiene k columnas, exigiremos:

$$Rango(\boldsymbol{x}) = k$$

### 2.1.6. Algunas Consecuencias del Modelo de Regresión Clásico

a) Definición de matriz de varianza-covarianza para $\widehat{\boldsymbol{\beta}}$:

$$E[(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta})(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta})']$$

Usando la expresión de$\widehat{\boldsymbol{\beta}}$ se prueba (Gujarati y Porter, 2010, p.875):

$$E[(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta})(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta})'] = \sigma^2(\boldsymbol{x'x})^{-1}$$

b) Además, $\widehat{\boldsymbol{\beta}}$ es un estimador lineal puesto que $(\boldsymbol{x'x})^{-1}\boldsymbol{x'}$ es una matriz conteniendo valores constantes y tiene la forma:

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{x'x})^{-1}\boldsymbol{x'y}$$

Puesto que $\boldsymbol{y} = \boldsymbol{x\beta} + \boldsymbol{\mu}$:

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{x'x})^{-1}\boldsymbol{x'}(\boldsymbol{x\beta} + \boldsymbol{\mu})$$

Como $(\boldsymbol{x'x})^{-1}(\boldsymbol{x'x}) = I$ se tiene:

$$\widehat{\boldsymbol{\beta}} = \boldsymbol{\beta} + (\boldsymbol{x'x})^{-1}\boldsymbol{x'\mu} \rightarrow E(\widehat{\boldsymbol{\beta}}) = \boldsymbol{\beta} + (\boldsymbol{x'x})^{-1}\boldsymbol{x'}E(\boldsymbol{\mu}) \rightarrow$$

$$E(\widehat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$$

Luego, $\widehat{\boldsymbol{\beta}}$ es un estimador "insesgado". También se puede probar (Gujarati y Porter, 2010, p.95) que es el de menor varianza entre los estimadores lineales (best linear unbiased estimator, BLUE.) Eso constituye el teorema de Gauss-Markov:

En el modelo de regresión lineal clásico, los estimadores de mínimos cuadrados $\widehat{\boldsymbol{\beta}}$ son estimadores lineales insesgados con varianza mínima (BLUE.)

### 2.1.7. Interpolación de Polinomios

El método de mínimos cuadrados se puede emplear satisfactoriamente para ajustar un conjunto de datos mediante un polinomio cuyo grado es inferior al número de muestras (Lancaster y Salkauskas, 1986, p.55):

Si las muestras son los pares $(x_0, f_0),\ (x_1, f_1)...(x_N, f_N)$ que se aproximan mediante el polinomio $p(x) = a_0 + a_1x + a_2x^2 + ...a_mx^m$, la aplicación del principio de mínimos cuadrados genera las ecuaciones:

$$a_0\sum_{i=0}^{N} x_i^0 + a_1\sum_{i=0}^{N} x_i + ... + a_m\sum_{i=0}^{N} x_i^m = \sum_{i=0}^{N} f_i$$

$$a_0 \sum_{i=0}^{N} x_i + a_1 \sum_{i=0}^{N} x_i^2 + ... + a_m \sum_{i=0}^{N} x_i^{m+1} = \sum_{i=0}^{N} x_i f_i$$

...

$$a_0 \sum_{i=0}^{N} x_i^m + a_1 \sum_{i=0}^{N} x_i^{m+1} + ... + a_m \sum_{i=0}^{N} x_i^{2m} = \sum_{i=0}^{N} x_i^m f_i$$

las cuales se pueden reescribir en forma matricial considerando una matriz de Vandermonde conteniendo las potencias de los componentes independientes de las muestras, las incógnitas y las componentes dependientes de las muestras:

$$\boldsymbol{V} = \begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^m \\ 1 & x_1 & x_1^2 & ... & x_1^m \\ & & & ... & \\ 1 & x_N & x_N^2 & ... & x_N^m \end{bmatrix} \; ; \; \boldsymbol{a} = \begin{bmatrix} a_0 \\ a_1 \\ ... \\ a_m \end{bmatrix} \; ; \; \boldsymbol{f} = \begin{bmatrix} f_0 \\ f_1 \\ ... \\ f_m \end{bmatrix}$$

mediante:

$$\boldsymbol{V}^T \boldsymbol{V} \boldsymbol{a} = \boldsymbol{V}^T \boldsymbol{f}$$

Luego, asumiendo que $\boldsymbol{V}$ es invertible, se obtiene la solución para los coeficientes del polinomio:

$$\boldsymbol{a} = \boldsymbol{V}^{-1} \boldsymbol{f} \tag{2.1.3}$$

## 2.2. Distribución de Poisson y Procesos de Poisson

En esta sección realizaremos una deducción teórica de este proceso desde los principios básicos de la teoría de probabilidades.

### 2.2.1. Distribución Binomial

Dada una secuencia de $n$ experimentos independientes cuyo resultado consiste en dos posibilidades (éxito/fracaso), donde la probabilidad de éxito viene dada por $p$ y la de fracaso por $(1-p)$, la distribución binomial se corresponde a la la distribución (discreta) de probabilidades del número total de éxitos. Evidentemente, el número de éxitos dado por $k$ varía en el rango de cero a $n$.

Cada conjunto de experimentos que conducen a $k$ éxitos consiste a su vez de $(n-k)$ fracasos. Siendo estos experimentos independientes, la probabilidad de cada conjunto se obtiene multiplicando las probabilidades individuales, las que totalizan:

$$p^k(1-p)^{n-k}$$

Asimismo, los $k$ éxitos pueden distribuirse de $\binom{n}{k}$ formas distintas, cada una de las cuales con la probabilidad señalada, por lo cual se alcanza una probabilidad total de:

$$p(X=k) = \binom{n}{k} p^k(1-p)^{n-k}$$

### 2.2.2. Procesos de Conteo

"A stochastic process $\{N(t), t \geq 0\}$ is said to be a counting process if N (t) represents the total number of "events" that occur by time t ." [Un proceso estocástico $\{N(t), t \geq 0\}$ es un proceso de conteo si $N(t)$ representa el número total de "eventos" que ocurren hasta el tiempo $t$.] (Ross, 2014)

Así, un proceso de conteo satisface:

1. $N(t) \geq 0$
2. $N(t) \in \mathbb{Z}$
3. $s < t \Rightarrow N(s) \leq N(t)$
4. Si $s < t$ entonces la diferencia $N(t) - N(s)$ corresponde al número de eventos que ocurren en el intervalo temporal $(s, t]$.

A modo de ejemplo, si decimos que un evento ocurre cada vez que nace un niño, entonces $\{N(t), t \geq 0\}$ es un proceso de conteo donde $N(t)$ es igual al número total de personas nacidas hasta el tiempo $t$.

Una propiedad complementaria muy usual corresponde a la independencia de los eventos ocurridos en intervalos disjuntos (denominada "independencia de incrementos"); esta asunción puede ser válida o aproximadamente válida de acuerdo a las circunstancias concretas que se está modelando. En el ejemplo de las personas nacidas, es razonable pensar que el número de nacimientos totales ocurridos durante el año 2017 en el Perú no sea determinante para el número de nacimientos ocurridos durante 2018 en este mismo país (por lo que se reportarían valores similares); sin embargo, el número de nacimientos (considerado excesivo) de la

década de los sesenta en China sí fue determinante para imponer las famosas políticas de "sólo un hijo" de fines de los setenta, con el consiguiente decrecimiento de las tasas de natalidad.

Asimismo un proceso se dice que posee "incrementos estacionarios" si el número de eventos en el intervalo $(s,\ s+\Delta)$ no depende de $s$; es decir, sólo depende del tamaño del intervalo mas no del momento en que se realiza el experimento.

### 2.2.3. Proceso de Poisson

El proceso de Poisson es un proceso de conteo ideal en el cual los eventos son independientes unos de otros. En este escenario ocurren eventos (de duración cero) en forma aleatoria pero con una tasa constante; es decir, existe una constante $\lambda$ a la que tiende la razón del número de eventos que ocurren para una unidad de tiempo considerada.

Por ejemplo, en un "call center" de atención a clientes se puede recibir en promedio 450 llamadas por hora; la independencia se refleja en tanto cada uno de los clientes hace su llamada sin conocer o interesarse por las llamadas que otros clientes realizan, por lo que la tasa de llamadas permanece (bajo cierta tolerancia) relativamente constante.

La distribución de Poisson corresponde a la probabilidad de ocurrencia de $k \in \mathbb{N} \cup \{0\}$ eventos en la unidad de tiempo considerada. A continuación deduciremos la fórmula de la distribución de probabilidad de Poisson como un escenario límite de la distribución binomial.

Considérese que la unidad de tiempo está dividida en $n$ subintervalos iguales ($n > \lambda$). Cada subintervalo es susceptible de generar un evento (éxito) o no generarlo (fracaso.) Puesto que en promedio esperamos $\lambda$ eventos (éxitos) en la unidad de tiempo, entonces la probabilidad de éxito de un subintervalo será $\frac{\lambda}{n}$. Luego, para esta misma unidad de tiempo, podemos aproximar la probabilidad de $k$ eventos mediante una distribución binomial con $p = \frac{\lambda}{n}$:

$$p(X = k) = \binom{n}{k} (\frac{\lambda}{n})^k (1 - \frac{\lambda}{n})^{n-k}$$

Los eventos del proceso de Poisson son instantáneos, por lo que el último modelo debe ser llevado al límite $n \to \infty$, En tal caso empleamos la aproximación del Stirling:

$$n! \approx \sqrt{2\pi n} (\frac{n}{e})^n \Rightarrow$$

$$p(X = k) = \frac{n!}{k!(n-k)!} (\frac{\lambda}{n})^k (1 - \frac{\lambda}{n})^{n-k} =$$

$$\frac{\sqrt{2\pi n}(\frac{n}{e})^n}{k!\sqrt{2\pi(n-k)}(\frac{n-k}{e})^{n-k}}(\frac{\lambda}{n})^k(1-\frac{\lambda}{n})^{n-k} = \frac{1}{k!}\sqrt{\frac{n}{n-k}}e^{-k}\lambda^k(\frac{n-\lambda}{n-k})^{n-k}$$

analizando el último paréntesis:

$$(\frac{n-\lambda}{n-k})^{n-k} = (\frac{n-k+k-\lambda}{n-k})^{n-k} = (1+\frac{k-\lambda}{n-k})^{n-k}$$

Utilizando la identidad: $lim_{x\to\infty}(1+\frac{1}{x})^x = e$, se tiene:

$$= [(1+\frac{1}{(\frac{n-k}{k-\lambda})})^{\frac{n-k}{k-\lambda}}]^{k-\lambda} = [e]^{k-\lambda}$$

Reemplazando:

$$p(X=k) = \frac{1}{k!}\sqrt{\frac{n}{n-k}}e^{-k}\lambda^k(e^{k-\lambda})$$

Haciendo $n \to \infty$ el radical tiende a la unidad, por lo que tenemos la fórmula conocida para la distribución de Poisson con tasa $\lambda$:

$$p(X=k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Para un intervalo arbitrario de longitud $t$ se espera un número de eventos $\lambda t$ pues la tasa es constante. En tal caso la probabilidad de $k$ eventos en este intervalo es

$$p(X=k) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}$$

De la última expresión se obtiene la media de la distribución:

$$E[X] = \sum_{k=0}^{\infty} k.p(X=k) = e^{-\lambda t}\sum_{k=0}^{\infty} k\frac{(\lambda t)^k}{k!}$$

Descartando el término para $k=0$ (por ser igual a cero):

$$E[X] = e^{-\lambda t}\sum_{k=1}^{\infty}\frac{(\lambda t)^k}{(k-1)!}$$

De otro lado $\sum \frac{x^k}{k!} = e^x \Rightarrow \sum \frac{x^{k-1}}{(k-1)!} = e^x \Rightarrow x\sum \frac{x^{k-1}}{(k-1)!} = xe^x \Rightarrow \sum \frac{x^k}{(k-1)!} = xe^x$, luego:

$$E[X] = e^{-\lambda t}(\lambda t)e^{\lambda t} = \lambda t$$

En forma similar se verifica que la varianza también es igual a $\lambda t$.

**Definición.** El proceso de conteo $\{N(t), t \geq 0\}$ es de Poisson con tasa (o intensidad) $\lambda$ si satisface (Ross, 2014, p.304):

1. $N(0) = 0$
2. Los incrementos de $N(t)$ son independientes
3. El número de eventos en un intervalo de longitud $s$ sigue la distribución de Poisson con media $\lambda s$, esto es:

$$P(N(t+s) - N(t) = k) = \frac{(\lambda s)^k e^{-\lambda s}}{k!} \tag{2.2.1}$$

### 2.2.4. Proceso de Poisson no Homogéneo (NHPP)

Consiste en una generalización del proceso de Poisson anteriormente descrito. El proceso de conteo $\{N(t), t \geq 0\}$ es de Poisson no homogéneo cuando su tasa $\lambda$ varía a través del tiempo, lo que se denomina "función de intensidad" $\lambda(t)$.

El principal atractivo de del proceso de Poisson no homogéneo consiste en que no exige incrementos estacionarios; es decir, los eventos pueden ocurrir con distinta tasa en distintos momentos. Un ejemplo típico es el "call center" antes mencionado: la tasa de llamadas en horas de la madrugada suele ser una pequeña fracción en relación a las horas "laborables" por lo que un proceso de Poisson simple (homogéneo) resulta insuficiente para su modelamiento.

**Definición.** El valor esperado de $N(t)$ se denomina función de valor medio (Ross, 2014, p.333):

$$E[N(t)] = m(t) = \int_0^t \lambda(s)ds \tag{2.2.2}$$

Lo que permite obtener una expresión para la probabilidad de $k$ eventos en el intervalo $[0, t)$[1]:

$$p(N(t) = k) = \frac{m(t)^k e^{-m(t)}}{k!} \tag{2.2.3}$$

## 2.3. Estimación de Parámetros para NHPP

En esta sección se revisa el procedimiento usual para estimar los parámetros de una distribución de la familia exponencial (como es el caso de la distribución

[1]Esta expresión generaliza (2.2.1) pero a la vez la restringe al escenario $[0, t)$; el intevalo $[t, t + s)$ sufre las substituciones formales $t \to 0$ y $s \to t$.

exponencial, la distribución de Poisson, la distribución NHPP, entre otros.) Siguiendo a Pham (2007, p.79), sea la función de densidad de probabilidades $f(x, \theta)$ con parámetros de a estimar $\theta_1, \theta_2 ... \theta_m$, a partir de datos disponibles correspondientes a observaciones independientes $X_1, X_2 ... X_n$ que se suponen siguen dicha distribución, entonces la densidad de probabilidad asociada (probabilidad de observar) a una muestra $X_i$ es por definición $f(X_i, \theta) = f(X_i, \theta_1 ... \theta_m)$. De aquí, la probabilidad de observar precisamente el conjunto $\{X_1, X_2 ... X_n\}$ es

$$f(\boldsymbol{X}, \theta) = f(X_1, \theta) f(X_2, \theta) ... f(X_n, \theta)$$

El uso del término "probabilidad" convencionalmente supone conocidos los parámetros $\theta$; como este no es el caso, diremos que esto corresponde a la "verosimilitud" (likelihood) de que para ciertos parámetros $\theta$ se observen las muestras $x_i$. El problema se traduce ahora en hallar las variables (originalmente parámetros) $\theta$ que hacen más verosímil tales muestras, lo que en términos matemáticos se convierte en un problema de maximización de la función $f(\theta)$ (con $X_i$ constantes.)

Usualmente (y en especial para las distribuciones de la familia exponencial) es más conveniente maximizar la función $\log(f(\theta))$, lo cual es equivalente al problema anterior dado que $\log(.)$ es estrictamente creciente en $\mathbb{R}^+$. Esto significa que la expresión a maximizar es:

$$L(X, \theta) = \log(\prod_{i=1}^{n} f(X_i, \theta)) = \sum_{i=1}^{n} \log(f(X_i, \theta)) \tag{2.3.1}$$

El "estimador" $\widehat{\theta} = h(\boldsymbol{X})$ que maximiza la expresión anterior se denomina "estimador de máxima verosimilitud". De acuerdo a 2.6.3, se encuentra resolviendo el sistema de $m$ ecuaciones:

$$\frac{\partial L(X, \theta)}{\theta_i} = 0 \tag{2.3.2}$$

A continuación aplicaremos este procedimiento para estimar los parámetros del modelo de Goel-Okumoto, representante de la clase de modelos basados en NHPP.

Siguiendo la ecuación 2.2.2, el número de eventos $X_i$ entre entre dos instantes $t_{i-1}$ y $t_i$ tiene por valor medio:

$$E[X_i] = E[N(t_i) - N(t_{i-1})] = m(t_i) - m(t_{i-1})$$

Lo anterior es válido para $i > 1$; pero como $E[X_i] = m(t_i)$ la ecuación anterior es válida para $i \geq 1$ si consieramos $t_0 = 0$ y $m(t_0) = m(0) = 0$.

Si estos eventos son variables aleatorias mutuamente independientes nos ha-

llaremos ante un NHPP, donde su densidad de probabilidad viene dada por

$$P(X_i = k) = \frac{e^{-(m(t_i)-m(t_{i-1}))}[m(t_i) - m(t_{i-1})]^k}{k!}$$

por lo que la función logarítmica de verosimilitud es:

$$L(X,\theta) = \sum_{i=1}^{n} \log(f(X_i,\theta)) = \sum_{i=1}^{n} \log(\frac{e^{-(m(t_i)-m(t_{i-1}))}[m(t_i) - m(t_{i-1})]^{X_i}}{X_i!}) =$$

Por propiedad telescópica:

$$\sum_{i=1}^{n}[m(t_i) - m(t_{i-1})] = m(t_n) - m(t_0)$$

$$\sum_{i=1}^{n} X_i \log(m(t_i) - m(t_{i-1})) - [m(t_n) - m(t_0)] - \sum_{i=1}^{n} \log(X_i!) \tag{2.3.3}$$

### 2.3.1. Estimación para modelo Goel-Okumoto

Usando la ecuación 2.2.3 que determina el modelo Goel-Okumoto:

$$m(t) = a(1 - e^{-bt}) \Rightarrow$$

$$m(t_i) - m(t_{i-1}) = -a(e^{-bt_i} - e^{-bt_{i-1}})$$

$$m(t_n) - m(t_0) = a(1 - e^{-bt_n})$$

Reemplazando en 2.3.3:

$$L(X,\theta) = \sum_{i=1}^{n}[X_i \log(m(t_i) - m(t_{i-1}))] - a(1 - e^{-bt_n}) - \sum_{i=1}^{n} \log(X_i!)$$

Calculando las derivadas parciales:

$$\frac{\partial L(X,\theta)}{\partial a} = \sum_{i=1}^{n} \frac{X_i}{m(t_i) - m(t_{i-1})} \frac{\partial[m(t_i) - m(t_{i-1})]}{\partial a} - (1 - e^{-bt_n}) =$$

$$\sum_{i=1}^{n} \frac{X_i}{-a(e^{-bt_i} - e^{-bt_{i-1}})}[-(e^{-bt_i} - e^{-bt_{i-1}})] - (1 - e^{-bt_n}) =$$

$$\frac{\sum_{i=1}^{n} X_i}{a} - (1 - e^{-bt_n})$$

La suma $\sum_{i=1}^{n} X_i$ es el número total (acumulado) de fallos en $t_n$, es decir $N(t_n)$; luego, igualando a cero:

$$\frac{N(t_n)}{a} = 1 - e^{-bt_n} \tag{2.3.4}$$

De otro lado:

$$\frac{\partial L(X,\theta)}{\partial b} = \sum_{i=1}^{n} \frac{X_i}{m(t_i) - m(t_{i-1})} \frac{\partial [m(t_i) - m(t_{i-1})]}{\partial b} - at_n e^{-bt_n} =$$

$$\sum_{i=1}^{n} \frac{X_i (t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{-(e^{-bt_i} - e^{-bt_{i-1}})} - at_n e^{-bt_n}$$

Igualando a cero:

$$at_n e^{-bt_n} + \sum_{i=1}^{n} \frac{X_i (t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{e^{-bt_i} - e^{-bt_{i-1}}} = 0 \tag{2.3.5}$$

El sistema de ecuaciones formado por 2.3.4 y 2.3.5 permite resolver $a$ y $b$ mediante métodos numéricos, lo que constituye la estimación para el modelo Goel-Okumoto.

### 2.3.2. Contrastación de Modelos Estadísticos

Un problema frecuente en el análisis de datos empíricos consiste en la adecuación de éstos a un modelo matemático. En muchos casos es conveniente optar por un enfoque probabilístico que permita analizar la bondad de un modelo cuyos parámetros han sido ajustados (en la medida de lo posible) a los datos empíricos disponibles, tal como hemos observado en las últimas secciones.

En ocasiones se dispone de más de un modelo (que se ajusta a los mismos datos empíricos), por lo que surge la pregunta acerca de cuál modelo es el que permite un "mejor ajuste" de los datos. Si se asume que uno de estos modelos es el que realmente ocurre en la realidad, entonces es de esperarse que éste permita un mejor ajuste de los datos, especialmente en la medida que el volúmen de estos últimos crezca.

Sin embargo, es sabido que un los modelos pueden ajustar prácticamente cualquier conjunto de datos en forma trivial en la medida que el número de parámetros crezca acorde con la cantidad de datos a ajustar (el ejemplo ilustrativo típico sobre este punto es la capacidad de un polinomio de grado $n$ de ajustar perfectamente $n + 1$ datos de muestra.) En ese sentido, los investigadores han

buscado criterios complementarios para seleccionar modelos haciendo un balance entre la capacidad de ajuste a los datos versus la redundancia de los parámetros.

Al respecto, Ding, Tarokh y Yang (2018) proporciona una revisión actualizada de los métodos más empleados para contrastar el poder de adecuación de modelos a los conjuntos de datos disponibles, observándose que tales modelos tienden a ser superiores que sus pares en ciertos escenarios, pero en otros les ocurre todo lo contrario. En especial, ciertos métodos son más adecuados para el problema de encontrar el modelo que proporcione el mejor ajuste, mientras que otros tienden a ser preferibles cuando se trata de hacer predicciones.

Uno de los métodos más populares corresponde al "criterio de información de Akaike" (Akaike, 1974), usualmente abreviado como AIC, cuya fundamentación proviene de ideas de la teoría de la información[2]. Si $N$ es el número de parámetros y $MLE$ es el máximo de la función logaritmo de verosimilitud para las muestras, entonces:

$$AIC = -2.log(MLE) + 2N$$

Tener en cuenta que los criterios basados en la teoría de la información como AIC, no son un test en el sentido usual de la inferencia estadística, y no hay conceptos asociados como potencia de prueba o niveles de significancia. Los tests de hipótesis estadísticos representan un paradigma muy diferente, y generalmente inferior, para el análsis de datos en contextos complejos (Anderson y Burnham, 2002, p.84), especialmente en lo que corresponde a la selección de modelos. De otro lado, existen reglas prácticas en relación a las diferencias entre los valores de $AIC$ obtenidos para un conjunto de modelos (denotados por $AIC_i$); denotando por $AIC_{min}$ al mínimo AIC de éstos, y a las respectivas diferencias $\Delta_i = AIC_i - AIC_{min}$, entonces la tabla 2.1 describe el nivel de "soporte empírico" proporcionado por los datos para el modelo "$i$".

## 2.4. Conceptos de Teoría de Fiabilidad

Esta sección es un extracto del apéndice B.2 de Lyu y col. (1996). Sea $T$ una variable aleatoria que representa el "tiempo hasta el fallo". $P(t \leq T \leq t + \Delta t)$ es

[2]Este autor señala que las iniciales "IC" corresponden a "information criteria"; siendo ésta la primera estimación de su clase, era bautizada como "AIC", la cual debería ser sucedida por "BIC", "CIC", etcétera (Akaike, 1974, p.719). Su principal componente es la minimización de la "distancia Kullback-Leibler", cuyo valor esperado se estima mediante la maximización de la función logaritmo de verosimilitud, con un sesgo asintóticamente coincidente con el número de parámetros (Anderson y Burnham, 2002, p.61). Como se aprecia, AIC incorpora el número de parámetros del modelo en sentido contrario a la verosimilitud, lo que castiga la excesiva complejidad de modelos con parámetros redundantes.

| $\Delta_i$ | Nivel |
|---|---|
| 0-2 | Substancial |
| 4-7 | Considerablemente reducido |
| >10 | Esencialmente ninguno |

Cuadro 2.1: Nivel de soporte empírico de modelos según Anderson y Burnham (2002, p.70).

la probabilidad de que el fallo ocurra en un intervalo temporal $[t, t + \Delta t]$. Esta probabilidad se modela mediante una función de densidad (de fallos) $f(t)$ cuya integral se representa por $F(t)$, así tenemos:

$$P(t \leq T \leq t + \Delta t) = F(t + \Delta t) - F(t) \approx f(t)\Delta t$$

Si el tiempo $t = 0$ entonces

$$F(t) = P(0 \leq T \leq t) = \int_0^t f(s)ds$$

**Definición.** La probabilidad de éxito (no fallo) en el tiempo $t$ denotada por $R(t)$ es la probabilidad de que el tiempo para fallo es mayor que $t$:

$$R(t) = P(T > t) = 1 - F(t) = \int_t^\infty f(s)ds$$

Anotemos que $F(0) = 0$ y $lim_{t\to\infty}F(t) = 1$ por los axiomas de la teoría de probabilidades.

Otro concepto útil corresponde a la tasa de fallo (failure rate.) Es definido como la probabilidad de ocurrencia del fallo un un intervalo $[t, t+\Delta t]$, condicionado a que el fallo no ha ocurrido antes de $t$:

$$Tasa\ Fallo = \frac{P(t \leq T < t + \Delta t | T > t)}{\Delta t} =$$

$$\frac{P(t \leq T < t + \Delta t)}{P(T > t)\Delta t} = \frac{F(t + \Delta t) - F(t)}{R(t)\Delta t}$$

**Definición.** La tasa de fallo instantánea se denomina "tasa de peligro" (hazard rate) y se denota por $z(t)$:

$$z(t) = lim_{\Delta t\to 0}\frac{F(t + \Delta t) - F(t)}{R(t)\Delta t} = \frac{f(t)}{R(t)}$$

La cantidad $z(t)dt$ representa la probabilidad de que un sistema de edad $t$ falle en el pequeño intervalo $[t, t + dt]$.

De la última expresión:

$$z(t) = f(t)\frac{1}{R(t)} = \frac{dF(t)}{dt}\frac{1}{R(t)}$$

como $R(t) = 1 - F(t) \Rightarrow \frac{dF}{dt} = -\frac{dR}{dt} \Rightarrow$

$$z(t) = -\frac{dR(t)}{dt}\frac{1}{R(t)} \Rightarrow -z(t)dt = \frac{dR(t)}{R(t)}$$

Integrando:

$$-\int_0^t z(s)ds + c = ln(R(t))$$

Dado que al inicio el sistema no ha fallado, $R(0) = 1 \Rightarrow c = 0 \Rightarrow$

$$R(t) = exp[-\int_0^t z(s)ds]$$

Además derivando:

$$\frac{dR}{dt} = -z(t)exp[-\int_0^t z(s)ds]$$

Pero $\frac{dR}{dt} = -\frac{dF}{dt} = -f \Rightarrow$

$$f(t) = z(t)exp[-\int_0^t z(s)ds]$$

A fin de que esta subsección sea autocontenida, enunciaremos el concepto de tiempo medio para fallo (Mean Time to Failure o MMTF.) Este indicador proporciona un valor que es fácilmente percibido por los usuarios del sistema, y se puede interpretar como la "esperanza de vida" del mismo, o el tiempo esperado durante el cual el sistema operará sin fallar.

**Definición.** El tiempo medio para fallo se define como:

$$MMTF = E[T] = \int_0^\infty tf(t)dt$$

Considerando una vez más que $f = F' = -R'$ se puede proporcionar una fórmula alternativa mediante integración por partes:

$$MMTF = -\int_0^\infty tR'dt = -tR|_0^\infty + \int_0^\infty t'Rdt = -tR|_0^\infty + \int_0^\infty Rdt$$

El primer término resultante requiere evaluar la expresión indeterminada ($\infty$,0):

$$lim_{t\to\infty} - tR(t)$$

Sin embargo, en los modelos de fiabilidad considerados $R(t)$ tiene un decaimiento exponencial, lo que hace que este límite sea cero. Luego, se tiene la expresión alternativa:

$$MMTF = \int_0^\infty Rdt$$

### 2.4.1. Ocurrencia de fallos en el proceso de Poisson

Sea $M(t)$ el número de fallos ocurridos hasta el tiempo $t$, con función de valor medio $m(t)$:

$$m(t) = E[M(t)]$$

El número de fallos para entre dos instantes $t_{i-1}$ y $t_i$ es definido como $f_i$ y su valor medio es:

$$E[f_i] = E[M(t_i) - M(t_{i-1})] = m(t_i) - m(t_{i-1})$$

Si $m(t)$ es una función lineal de la forma $\lambda t$ ($\lambda > 0$) entonces estamos ante un proceso de Poisson clásico (homogéneo), y de lo contrario ante un proceso de Poisson no homogéneo.

La intensidad de fallos se define como la derivada de $m(t)$:

$$\lambda(t) = \frac{dm(t)}{dt} \tag{2.4.1}$$

Para un incremento en la fiabilidad de un sistema, debe observarse que para algún tiempo $t_0$:

$$\forall t \geq t_0 : \frac{d\lambda(t)}{dt} < 0$$

## 2.5. Promedio Móvil

El Promedio Móvil (Moving Average) es una técnica típicamente aplicada para "suavizar" las series de tiempo con fluctuaciones; esto se hace tanto con el fin de descubrir las tendencias de largo plazo del fenómeno observado, como para borrar la tendencia y estudiar el componente oscilatorio (generalmente estacional.) Originalmente estudiada en la ciencia actuarial, es empleada en la actualidad en diversas disciplinas, notablemente en el estudio de las tendencias de los mercados:

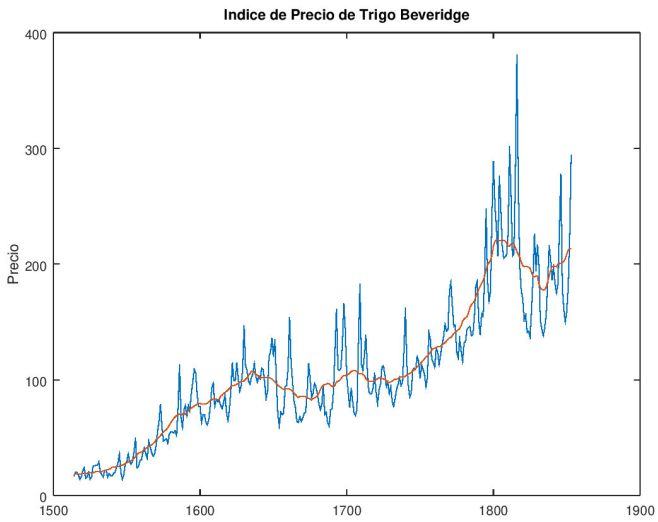



Figura 2.5.1: Índice Beveridge y Promedio Móvil.
Precios del trigo en línea azul y promedio móvil en línea roja a través de los años; el promedio móvil está centrado, y su orden es de 31 años.

> Un promedio móvil es simplemente el promedio aritmético de un número determinado de anteriores precios de una acción o de un índice. (Siegel, 2008, p.295)

A modo de ejemplo, ver la figura 2.5.1 donde se el promedio móvil permite apreciar una tendencia de largo plazo en un conocido índice de precios del trigo europeo[3].

Más formalmente, dada una secuencia $p_1, p_2, ...$, el promedio móvil corresponde a una secuencia donde el término i-ésimo se obtiene mediante:

$$\overline{p_i} = \frac{1}{n}(p_i + p_{i-1} + .. + p_{i-[n-1]})$$

También es usual emplear una versión "centrada" (especialmente si el interés no es hacer predicciones):

$$\overline{p_i}_C = \frac{1}{2k+1}(p_{i-k} + ... + p_{i-1} + p_i + p_{i+1} + ... + p_{i+k})$$

Es usual que a este promedio móvil se le denomine "simple" en contrapartida con el promedio móvil ponderado (weighted moving average), que generalmente reduce el factor de ponderación conforme los términos se alejan en el pasado con la idea de que éstos van perdiendo importancia conforme son más antiguos:

$$\overline{p_i}_W = \alpha_1 p_i + \alpha_2 p_{i-1} + .. + \alpha_n p_{i-[n-1]}\ ;\ \sum \alpha_i = 1$$

[3]El índice Beveridge es empleado recurrentemente en el análisis económico de series de tiempo y fue publicado en 1921 Mills (2011, p.26).

Nuestro interés en el promedio móvil apunta a eliminar la variación aparentemente caótica en el registro de las líneas de código fuente a escala diaria, con la asunción implícita de existir una tendencia subyacente que puede ser descubierta en el promedio móvil de cierto número de días.

Si bien existe cierto nivel de relación, esta técnica no debe confundirse con los procesos estocásticos de clase "MA" (moving average) que se estudian las series de tiempo, típicamente en conjunto con los de clase autoregresiva "AR" (Paolella, 2018, p.294).

## 2.6. Técnicas de Optimización

En esta sección se pasa revista a la técnica de optimización no lineal conocida como "sequential quadratic programming" (programación cuadrática secuencial.) Describimos esta técnica -entre otras disponibles- por un motivo estrictamente práctico: es empleada por el software matemático Octave a través de la función `sqp()` que utilizamos para el proceso de maximización de funciones logarítmicas de verosimilitud.

La técnica de programación cuadrática secuencial se basa en un conjunto de resultados que pasamos a reseñar de modo muy abreviado.

### 2.6.1. Problema general de Programación Lineal

Los problemas de programación lineal se presentan en numerosas disciplinas por lo que han sido ampliamente estudiados. En la forma canónica, su objetivo es maximizar (o minimizar) la expresión:

$$f(X) = c_1x_1 + c_2x_2 + ... + c_nx_n$$

sujeta a $m > 0$ restricciones:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &= b_2 \\ &... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n &= b_m \end{aligned}$$

donde las incógnitas son positivas:

$$x_i \geq 0 \; ; i = 1..n$$

Esto se puede reescribir convenientemente en forma matricial:

$$Maximizar: \ C^T X$$

sujeto a:

$$AX = B$$
$$X \geq 0$$

Aquí $A$ es una matriz de $m * n$ y $B$ es de $m * 1$; la última expresión denota la positividad de las incógnitas que constituyen el vector $X = (x_1...x_n)^T$.

En muchas ocasiones las restricciones se dan en forma de desigualdades; por ejemplo, para $AX \leq B$ es posible transformar las desigualdades en igualdades mediante la introducción de variables de holgura $Z = (z_1...z_n)^T$, reescribiendo las desigualdades mediante:

$$AX + Z = B$$
$$z_i \geq 0$$

Asimismo, en la forma canónica las variables $x_i$ deben ser positivas; de no ser el caso, cada veriable $x_j$ sin esta restricción se puede convertir en dos nuevas variables $x_j^+ \geq 0$ y $x_j^- \geq 0$ donde $x_j^+ = max(x_j, 0)$ y $x_j^- = -max(-x, 0)$; con esta definición se tiene la fórmula de reemplazo $x_j = x_j^+ - x_j^-$.

Las restricciones del problema consideradas en simultáneo determinan una región del espacio a la que denominamos "conjunto factible"; los puntos de este conjunto que maximizan (o minimizan) la expresión objetivo son la solución del problema.

En la programación lineal las restricciones son ecuaciones (o inecuaciones) lineales, las que determinan un conjunto factible convexo.

**Definición.** Dado un conjunto convexo $S \subset \mathbb{R}^n$, $x$ es un punto extremo de $S$ si no existen $a, b \in S$ y $0 < t < 1$ tal que $x = ta + (1 - t)b$. Intuitivamente esto se refiere a los vértices del conjunto $S$ cuyas caras son los hiperplanos conformados por las restricciones.

Un resultado fundamental para la programación lineal es el siguiente

**Teorema.** *Si el conjunto factible $S$ de un problema de programación lineal es acotado, entonces el óptimo (máximo o el mínimo) de la función objetivo es al-*

*canzado en un extremo de S.*[4]

Este teorema permite reducir el espacio de búsqueda de soluciones a la frontera, lo que es aprovechado por los algoritmos creados para tal fin (Foulds, 1981, p.19), en particular por Simplex[5].

### 2.6.2. Formas Cuadráticas

Para efectos de la optimización de formas cuadráticas definidas mediante:

$$f(X) = X^T H X$$

donde $X = (x_1...x_n)^T$ es una matriz $n * 1$ y $H$ una matriz $n * n$. Esta forma cuadrática es negativa definida si

$$\forall X \neq 0 \Rightarrow f(X) < 0$$

A fin de dilucidar si una matriz es negativa definida se puede emplear el siguiente:

**Lema.** *Una matriz cuadrada es negativa definida si y sólo si todos sus valores propios son negativos.*

Desde el punto de vista de la optimización de la forma cuadrática, la proposición esencial es la siguiente:

**Proposición.** *Sea $X^*$ un punto crítico de $f$, esto es: $\nabla f(X^*) = 0$, si la matriz hessiana $H(X^*)$ es negativa definida, entonces $X^*$ es un mínimo local de $f$.*

### 2.6.3. Condiciones para Extremos

Los teoremas enunciados a continuación provienen del cálculo diferencial en varias variables. $X$ denota un vector de $n$ componentes $(x_1...x_n)^T$ y $f : S \to \mathbb{R}$ es una región en un espacio euclideano n-dimensional.

**Teorema.** *Si $\frac{\partial f(X)}{\partial x_j}$ existe para todos los puntos $X \in S$ con $j = 1...n$, y si $f$ tiene un extremo local en $X^*$ en el interior de $S$, entonces*

$$\frac{\partial f(X^*)}{\partial x_j} = 0 \ ; \ j = 1...n \tag{2.6.1}$$

[4]Notar el óptimo también puede ser alcanzado en un punto no extremo. En programación lineal esto puede considerarse un caso degenerado.

[5]El algoritmo Simplex se describe ampliamente en los textos de optimización como los que citamos en esta sección.

Lo anterior señala una condición necesaria que satisfacen los extremos. El teorema siguiente lo complementa proporcionando una condición suficiente:

**Teorema.** *Si se verifican las condiciones 2.6.1 para un punto $X^*$ al interior de $S$, y si $H(X^*)$ es negativa definida, donde $H(X^*)$ es la matriz hessiana de $f$ evaluada en $X^*$, entonces $f$ tiene un máximo local en $X^*$.*

Ver la prueba de estos teoremas en Foulds (1981, p.270).

### 2.6.4. Sistemas de Ecuaciones No Lineales

En esta sección enunciamos el método de Newton para resolver iterativamente un sistema de ecuaciones no lineales.

Sea $r : \mathbb{R}^n \to \mathbb{R}^n$ una función vectorial, el problema consiste en resolver:

$$r(x) = \mathbf{0}$$

Una forma de justificar este método es mediante el Teorema de Taylor de primer orden para el caso multidimensional:

**Teorema.** *(Taylor) Suponiendo que $r : D \to \mathbb{R}^m$ es continuamente diferenciable en un conjunto abierto convexo $D \subset \mathbb{R}^n$ y que $x$ y $x + p$ son vectores en $D$. Entonces*

$$r(x + p) = r(x) + J(x)p + ||p||.E(x, p)$$

donde $E(x, p) \to 0$ cuando $p \to 0$, y $J$ es la matriz jacobiana para $r$ (Apostol, 1969, p.271).

El método de Newton busca iterativamente la anulación de $f(x + p)$ seleccionando la dirección más adecuada para el vector $p$ mediante la aproximación

$$r_{k+1} = r(x_k + p) \approx r(x_k) + J(x_k)p_k$$

Si $r(x_k + p) = 0$ entonces se procede a resolver

$$0 = r(x_k) + J(x_k)p_k \Rightarrow$$

$$J(x_k)p_k = -r(x_k) \tag{2.6.2}$$

donde la matriz jacobiana es no singular. A nivel teórico, esto implica la siguiente operación[6]:

[6]La inversión de la matriz jacobiana es una operación computacionalmente costosa, por lo

$$p_k = -J(x_k)^{-1} r(x_k) \tag{2.6.3}$$

Luego, el algoritmo es:

■ Elegir $x_0$

■ Para $k = 0, 1, 2...$ calcular $p_k$ resolviendo 2.6.2 y obtener el nuevo punto mediante

$$x_{k+1} \leftarrow x_k + p_k$$

### 2.6.5. Optimización con restricciones en forma de igualdades[7]

Este problema de maximización es el paso inicial hacia el objetivo de la presente sección. Se define así:

$$\begin{aligned} Maximizar: &\quad f(X) \\ Sujeto\ a: &\quad g_j(X) = 0\ ;\ j = 1...m \end{aligned} \tag{2.6.4}$$

donde $X = (x_1...x_n)^T$. Cuando las restricciones son ecuaciones, entonces la solución debe estar en la frontera del conjunto factible. En lo que sigue asumiremos que $f$ tiene segundas derivadas continuas.

Se busca obtener una expresión para las primeras derivadas de $f$ que satisfacen las restricciones del problema. Los puntos estacionarios potenciales son aquellos en los que

$$\frac{\partial f}{\partial x_i} = 0 \ i = 1...\text{n} \tag{2.6.5}$$

Sea $X$ un punto que satisface las restricciones de 2.6.4; si la región definida por estas ecuaciones consta de más de un punto, entonces existe al menos un punto $X + h$ en todo entorno de $X$ que también las satisface. Expandiendo $f$ y $g_j$ en serie de Taylor alrededor de $X$ se tiene que conforme $X + h$ se aproxima a $X$:

$$\begin{aligned} f(X+h) &\approx f(X) + \nabla f(X)^T h \\ g_j(X+h) &\approx g_j(X) + \nabla g_j(X)^T h \end{aligned}$$

que se opta por resolver el sistema equivalente $(x_{k+1} - x_k)J(x_k) = -r(x_k)$. De otro lado, la familia de algoritmos denominados "quasi-newton" produce resultados a "menor costo" y usualmente con mayor estabilidad, siendo quizá el algoritmo de Broyden (Burden, Faires y D, 2010, p.648) el más popular de este grupo.

[7] Esta sección y la subsiguiente han sido extraídas de la sección 7.4 de Foulds (1981) y algunas indicaciones de Nocedal y Wright (2006).

Es decir

$$\partial f(X) \approx \nabla f(X)^T \partial X$$
$$\partial g_j(X) \approx \nabla g_j(X)^T \partial X$$

Llevadas a su forma diferencial estas aproximaciones se transforman en las ecuaciones:

$$\nabla f(X)^T \partial X = \partial f(X)$$
$$\nabla g_j(X)^T \partial X = \partial g \; ; j = 1...m$$

Estas igualdades constituyen un sistema de $(m+1)$ ecuaciones con $(n+1)$ incógnitas $\partial x_1 ... \partial x_n, \partial f(X)$. Podemos asumir que $m < n$ (caso contrario las restricciones conformarían un conjunto factible conformado por un único punto o incluso ninguno.) En lo que sigue se asume que estas ecuaciones son linealmente independientes (de lo contrario, se procede a eliminar las variables redundantes hasta lograrlo.)

Redefiniendo el vector $X$ como:

$$X = (w_1 ... w_m, y_1, ... y_{n-m})^T$$

donde las $w_i$ se denominan "variables de estado" y las $y_j$ se denominan "variables de decisión". El anterior sistema de ecuaciones se reescribe así:

$$\sum_{i=1}^{m} \frac{\partial f(X)}{\partial w_i} \partial w_i + \sum_{i=1}^{n-m} \frac{\partial f(X)}{\partial y_i} \partial y_i = \partial f(X)$$

$$\sum_{i=1}^{m} \frac{\partial g_j(X)}{\partial w_i} \partial w_i + \sum_{i=1}^{n-m} \frac{\partial g_j(X)}{\partial y_i} \partial y_i = \partial g \; ; j = 1...m$$

Sea la matriz jacobiana:

$$J = \begin{pmatrix} \frac{\partial g_1}{\partial w_1} & \frac{\partial g_1}{\partial w_2} & \dots & \frac{\partial g_1}{\partial w_m} \\ \frac{\partial g_2}{\partial w_1} & \frac{\partial g_2}{\partial w_2} & \dots & \frac{\partial g_2}{\partial w_m} \\ \dots & \dots & \dots & \dots \\ \frac{\partial g_m}{\partial w_1} & \frac{\partial g_m}{\partial w_2} & \dots & \frac{\partial g_m}{\partial w_m} \end{pmatrix}$$

Esta matriz siempre puede hacerse no singular eligiendo adecuadamente las variables de estado y de decisión dado que asumimos que las ecuaciones iniciales son linealmente independientes.

Además se define la matriz de control:

$$C = \begin{pmatrix} \frac{\partial g_1}{\partial y_1} & \frac{\partial g_1}{\partial y_2} & \cdots & \frac{\partial g_1}{\partial y_{m-m}} \\ \frac{\partial g_2}{\partial y_1} & \frac{\partial g_2}{\partial y_2} & \cdots & \frac{\partial g_2}{\partial y_{n-m}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial g_m}{\partial y_1} & \frac{\partial g_m}{\partial y_2} & \cdots & \frac{\partial g_m}{\partial y_{n-m}} \end{pmatrix}$$

Y sean:

$$W = (w_1 ... w_m)^T$$

$$Y = (y_1, ... y_{n-m})^T$$

Ahora podemos reescribir el sistema de ecuaciones mediante:

$$\nabla_w f^T \partial W + \nabla_y f^T \partial Y = \partial f(W, Y) \tag{2.6.6}$$

$$J\partial W + C\partial Y = \partial g \tag{2.6.7}$$

Si nos encontramos en un máximo que satisface las restricciones de 2.6.4 entonces $\partial g = 0$ con lo cual 2.6.7 se reduce a

$$J\partial W + C\partial Y = 0$$

Multiplicando por $J^{-1}$:

$$\partial W = -J^{-1} C \partial Y$$

Substituyendo en 2.6.6:

$$\partial f(W, Y) = (-\nabla_w f^T J^{-1} C + \nabla_y f^T) \partial Y$$

lo que permite formar el "gradiente restringido" de $f$ con respecto a $Y$:

$$\nabla_y^c f = \frac{\partial f^c(W, Y)}{\partial^c Y} = \nabla_y f^T - \nabla_w f^T J^{-1} C$$

Este gradiente representa el cambio resultante en $f$ al perturbar las componentes $y_i$ en el conjunto factible. Si $X^*$ es un máximo del conjunto factible (en el sentido señalado en la sección 2.6.3) se tiene:

$$\nabla_y^c f(X^*) = 0$$

Retornando a las ecuaciones 2.6.6 y 2.6.7, al eliminar $dW$ se tiene:

$$J\partial f(W,Y) - \nabla_w f^T \partial g = \nabla_y f^T J\partial Y - \nabla_w f^T C\partial Y \Rightarrow$$

$$\partial f(W,Y) = \nabla_w f^T J^{-1}\partial g + \nabla_y f^T \partial Y - \nabla_w f^T J^{-1} C\partial Y$$

los últimos dos términos corresponden al gradiente restringido, que en el punto $X^*$ se anula; luego:

$$\partial f(W^*,Y^*) = \nabla_w f^T J^{-1}\partial g \tag{2.6.8}$$

Esto permite formar el vector:

$$\frac{\partial f(W^*,Y^*)}{\partial g} = \nabla_w f^T J^{-1}$$

Este vector representa la tasa de cambio de $f(X^*)$ cuando $g$ es perturbado. A continuación definimos los multiplicadores de Lagrange mediante:

$$\lambda = (\lambda_1 ... \lambda_m) = \frac{\partial f(W^*,Y^*)}{\partial g} = \nabla_w f^T J^{-1} \tag{2.6.9}$$

reemplazando en 2.6.8:

$$\partial f(W^*,Y^*) = \lambda\partial g \tag{2.6.10}$$

definimos el lagrangiano:

$$F(X,\lambda) = f - \lambda g$$

con lo cual las restricciones de 2.6.4 y 2.6.10 se pueden reescribir como

$$\frac{\partial F}{\partial \lambda_j} = 0 \ ; \ j = 1...m \tag{2.6.11}$$

$$\frac{\partial F}{\partial x_i} = 0 \ ; \ i = 1...n \tag{2.6.12}$$

Esto constituye un sistema de $(m+n)$ ecuaciones con el mismo número de incógnitas $\lambda_1..\lambda_m, x_1...x_n$. Notar que estas condiciones son necesarias para un punto estacionario $X^*$ (para la suficiencia, se puede aplicar el segundo teorema de la sección 2.6.3.)

### 2.6.6. Optimización con restricciones en forma de desigualdades

Este problema de maximización es un paso intermedio en el objetivo de esta sección. Se define del siguiente modo:

$$\begin{array}{rl} Maximizar: & f(X) \\ Sujeto\ a: & g_j(X) \leq 0\ ;\ j = 1...m \end{array} \tag{2.6.13}$$

Transformamos las condiciones de este problema introduciendo variables auxiliares $s_j$ de modo tal que se aquellas se convierten en:

$$g_j(X) + s_j = 0$$

$$s_j \geq 0$$

donde $j = 1...m$. Si excluimos las inecuaciones para $s_j$ (que serán controladas externamente), esto se convierte en un problema con condiciones de igualdad como el que se describió en la sección anterior. Es necesario anotar que cada restricción $g_j$ puede o no determinar el máximo $X^*$ dependiendo de la ubicación de este punto en la frontera de la región factible; cuando éste es el caso se tiene la frontera determinada por $g_j(X^*) = 0$ y $s_j = 0$.

Generando el lagrangiano:

$$F(X, \lambda) = f(X) - \sum_{j=1}^{m} \lambda_j (g_j(X) + s_j) \tag{2.6.14}$$

de la definición 2.6.9 se tiene $\lambda_i = \frac{\partial f}{\partial g_i}$.

Siendo los valores $\lambda_j$ los coeficientes que describe el cambio de $f$ cuando cambian las funciones de restricción $g$, estas últimas pueden perturbarse si se reemplaza las desigualdades $g_i(X) \leq 0$ por $g_i(X) \leq \epsilon$ (donde $\epsilon$ es un valor muy pequeño), pues el conjunto de soluciones factibles no es reducido (sino ligeramente ampliado.) Así, el óptimo $f(X^*)$ no será inferior al del problema 2.6.13, y por tanto

$$\frac{\partial \mathrm{f}}{\partial g_j} = \lambda_j \geq 0 \tag{2.6.15}$$

Siguiendo el método de Lagrange, para un máximo local $(X^*, s^*, \lambda^*)$ se tiene para 2.6.11:

$$\frac{\partial F(X^*)}{\partial X} = 0 \Rightarrow$$

$$\nabla f(X^*) - \sum_{j=1}^{m} \lambda_j^* \nabla g_j(X^*) = 0 \tag{2.6.16}$$

Esto también se puede expresar mediante:

$$\nabla f(X^*) - J_g(X^*)^T \lambda = 0$$

donde $J_g$ es una matriz jacobiana de las funciones $g$ y $\lambda$ es el vector que contiene los elementos $\lambda_j$.

Asimismo, para 2.6.12:

$$\frac{\partial F(X^*)}{\partial \lambda} = 0 \Rightarrow$$

$$g(X^*) + S^* = 0 \tag{2.6.17}$$

donde $S = (s_1 ... s_m)$.

A partir de 2.6.15 podemos analizar dos escenarios. Si $\lambda_j = 0$ entonces por 2.6.9 tenemos que $\frac{\partial f}{\partial g_j} > 0$, en cuyo caso la restricción $j$ participa de la determinación del máximo, por lo que $s_j = 0$; luego de 2.6.17 se tiene que $g_j(X^*) = 0$.

De otro lado, si $g_j(X^*) < 0$, por 2.6.17 se tiene que $s_j > 0$, y por tanto $\frac{\partial f}{\partial g_j} = 0$, luego $\lambda_j = 0$. Estos dos casos se pueden resumir en la expresión más compacta:

$$\lambda_j g_j(X^*) = 0 \ ; \ j = 1...m \tag{2.6.18}$$

Las condiciones 2.6.18 son de tipo complementario: significan que restricción $g_j$ está activa o $\lambda_j = 0$, o posiblemente ambas.

Estas condiciones se resumen en el siguiente

**Teorema.** *Las ecuaciones 2.6.16, 2.6.18, 2.6.15 y la restricción de 2.6.13 son condiciones necesarias para la existencia de un máximo local $X^*$ y se denomian condiciones de Kuhn-Tucker (o Karush–Kuhn–Tucker).*

Mediante el análisis de convexidad del espacio admisible de soluciones, se estable el siguiente teorema más general:

**Teorema.** *Si en el problema 2.6.13 se tiene que $f$ es cóncava y las $g_j$ son convexas para $j = 1...m$, y existe el máximo local $X^*$ que satisface las condiciones de Kuhn-Tucker, entonces $f$ tiene un máximo global en $X^*$*[8].

[8]Ver Foulds (1981, p.288) para una demostración.

### 2.6.7. Programación Cuadrática

El problema de maximización cuadrática se define como:

$$\begin{aligned} Maximizar: &\quad f(X) = C^T X + X^T D X \\ Sujeto\ a: &\quad AX \leq B\ ;\ X \geq 0 \end{aligned}$$

donde $X = (x_1 ... x_n)^T$ es el vector de las variables, $C$ y $B$ son vectores de parámetros reales de dimensiones $n * 1$ y $m * 1$ respectivamente, y $D$ es una matriz simétrica negativa definida.

Por la proposición de minimización de la sección 2.6.2, $f$ es una función estrictamente cóncava sujeta a condiciones lineales. Las restricciones pueden reescribirse en la forma

$$AX - B \leq 0 \tag{2.6.19}$$

$$-X \leq 0 \tag{2.6.20}$$

lo que consigue un problema en la forma dada en la sección anterior (problema 2.6.13.) Los multiplicadores de Lagrange para 2.6.19 serán denominados $\lambda_1 ... \lambda_m$ y los correspondientes a 2.6.20 mediante $\delta_1 ... \delta_n$.

Aplicando Kuhn-Tucker tenemos para 2.6.16:

$$\nabla(C^T X + X^T D X) - \sum_{j=1}^{m} \lambda_j \nabla(AX - B) - \sum_{j=1}^{n} \delta_j \nabla(-IX) = 0$$

Aplicando las reglas 2.1.2 y 2.1.1, recordando que $D$ es simétrica:

$$C + 2X^T D - \sum_{j=1}^{m} \lambda_j A^T - \sum_{j=1}^{n} \delta_j (-I^T) = 0 \Rightarrow$$

$$C + 2X^T D - \lambda A - \delta = 0$$

repitiendo 2.6.13:

$$\begin{aligned} AX - B &\leq 0 \\ -X &\leq 0 \end{aligned}$$

de 2.6.18:

$$\lambda_j(AX - B) = 0$$
$$\delta_j(-X) = 0$$

finalmente, de 2.6.15:

$$\lambda_i \geq 0$$
$$\delta_i \geq 0$$

Estas relaciones definen un problema de programación lineal. Como la función $f$ es estrictamente cóncava, la solución hallada será óptima para el problema original.

### 2.6.8. Programación Cuadrática Secuencial

En esta sección describimos la técnica de programación cuadrática secuencial que proporciona una alternativa práctica para la solución de los problemas de optimización no lineal y es implementada por la función `sqp()` de Octave. Este material ha sido adaptado de la sección 18.1 de Nocedal y Wright (2006).

Retornando al problema sin desigualdades 2.6.4:

$$\begin{aligned} Maximizar: &\quad f(X) \\ Sujeto\ a: &\quad g_j(X) = 0\ ;\ j = 1...m \end{aligned}$$

Puede probarse que para este escenario las condiciones de Kuhn-Tucker se reducen a la ecuación 2.6.16 y la restricción del problema; escribiendo esto en forma matricial:

$$R(x, \lambda) = \begin{bmatrix} \nabla f(x) - J_g(x)^T \lambda \\ g(x) \end{bmatrix} = \mathbf{0}$$

Asimismo, el lagrangiano es $F(x, \lambda) = f(x) - \lambda^T g(x)$.

La solución de este sistema de ecuaciones puede intentarse por el método de Newton descrito en la sección 2.6.4, lo que conlleva el cálculo del jacobiano de R con respecto a las variables $x$ y $\lambda$. Operando convenientemente, esta matriz se puede expresar así:

$$R'(x, \lambda) = \begin{bmatrix} \nabla^2_{xx} F(x, \lambda) & -J_g(x)^T \\ J_g(x) & 0 \end{bmatrix}$$

Así, el método de Newton generará iterativamente elementos $(x_k; \lambda_k)$ mediante

los vectores gradientes de la forma $(p_x; p_\lambda)$, los cuales corresponden a soluciones de la ecuación 2.6.2; para nuestro problema, esta ecuación se expresa así mediante $R'p_k = -R$:

$$\begin{bmatrix} \nabla^2_{xx}F & -J_g^T \\ J_g & 0 \end{bmatrix}_k \begin{bmatrix} p_x \\ p_\lambda \end{bmatrix}_k = \begin{bmatrix} -\nabla f + J_g^T\lambda \\ -g \end{bmatrix}_k \tag{2.6.21}$$

El problema ahora consiste en resolver el sistema 2.6.21 sin recurrir a la inversión de la matriz jacobiana $R'$. Al respecto, considérese el problema auxiliar de optimización (para las variables del vector $p$):

$$\begin{array}{cc} Maximizar: & f_k + \nabla f_k^T p + \frac{1}{2}p^T\nabla^2_{xx}F_k p \\ Sujeto\ a: & J_g p + g_k = 0 \end{array} \tag{2.6.22}$$

este problema puede resolverse mediante programación cuadrática como se vio en la sección anterior[9]. Sin embargo, sabemos que el extremo (solución) satisface las condiciones de Kuhn-Tucker que aquí son:

$$\begin{aligned} \nabla^2_{xx}F_k p_k + \nabla f_k - J_g^T l_k &= 0 \\ J_g p_k + g_k &= 0 \end{aligned}$$

las cuales definen una solución $(\boldsymbol{p}; \boldsymbol{l})$. Es claro que este sistema es el mismo que 2.6.21, por lo que su solución también le es aplicable: $p_x = \boldsymbol{p}$ y $p_\lambda = \boldsymbol{l}$, lo que proporciona la solución necesaria para la iteración del método de Newton.

Este método se extiende directamente al problema que contiene desigualdades en las restricciones adaptando el problema auxiliar 2.6.22; ver Nocedal y Wright (2006, p.533) para más detalles.

[9]Esto es así pues si se asume continuidad de las derivadas del lagrangiano entonces la matriz hessiana $\nabla^2_{xx}F_k$ es simétrica (Apostol, 1969, p.308).

# Capítulo 3

# Datos de Muestra y su Procesamiento

## 3.1. Diseño de Investigación

La presente investigación es de tipo correlacional, en la medida que intenta establecer la relación entre variables mediante análisis matemático y estadístico. La información utilizada corresponde a los productos generados en proyectos de desarrollo de software (aplicativos), la mayoría de los cuales actualmente está en operación; es decir, no se basa en datos simulados.

En primer término, esperamos que las conclusiones sean aplicables a proyectos de software con las mismas características tecnológicas que los que aquí se estudian, con las correspondientes adecuaciones a la metodología e idiosincrasia de cada organización interesada. Adicionalmente, esta investigación puede servir como referencia para el establecimiento de modelos análogos para otras situaciones tecnológicas y/o organizacionales; la necesidad de disponer de modelos alternativos se desprende de la antes mencionada inexistencia de un modelo conveniente para toda clase de escenario.

En las secciones subsiguientes describiremos el proceso de extracción de los datos correspondientes a un conjunto de proyectos, los cuales servirán para estudiar la calidad de los modelos matemáticos a ser propuestos posteriormente. Este estudio está basado en el nivel de ajuste del modelo a los datos disponibles, tal como es usual en el ámbito de los estudios de estimación de esfuerzo de proyectos.

Las "muestras" recogidas en esta sección corresponden al conteo de líneas de código fuente de los proyectos a lo largo del tiempo, lo que puede enunciarse matemáticamente mediante una función $N$:

$$N: [0, t_F] \subset \mathbb{Z} \to \mathbb{N} \cup \{0\}$$
$$t \mapsto N(t)$$

Más precisamente, se recogen los conteos de las líneas agregadas (que informalmente denotaremos como "$ADD$") y las líneas eliminadas ("$DEL$"), lo que da lugar a métricas de "código neto" ($ADD - DEL$) y "code chunk" ($ADD + DEL$.)

En tanto el dominio corresponde a enteros no negativos, es usual que se denote los valores de las muestras mediante un subíndice:

$$N_i = N(t_i)$$

Por comodidad se asume que el proyecto se inicia en $t_0 = 0$ y termina en un día (o semana según el caso) denotado por $t_F$.

## 3.2. Población de Estudio

### 3.2.1. Unidad de Análisis

La unidad de análisis corresponde a un conjunto de proyectos de desarrollo de software "hecho a medida" ejecutados por el autor y su equipo de desarrollo entre los años 2008 y 2014. Estos proyectos son de tipo multicapa con almacenamiento relacional, y utilizan lenguaje Java. Han sido desarrollados por encargo de dos importantes empresas peruanas. A partir de la población correspondiente a aproximadamente 100 proyectos de diversas características desarrollados por el autor, se ha seleccionado una muestra de siete de éstos, considerados principalmente por corresponder íntegramente al perfil señalado; es decir, otros proyectos de la población tienen (en diverso grado) rasgos adicionales o componentes complementarios que no encajan con el perfil mencionado (por ejemplo, otros lenguajes de programación, actividades relacionadas con el sistema operativo, programas para ejecución en lotes, etc.)

Por cuestiones de confidencialidad, los proyectos señalados serán denotados mediante una codificación neutral de dos caracteres; éstos se enumeran en las tablas 3.1 y 3.2.

A continuación se proporciona una descripción superficial de los proyectos y se realizan algunos comentarios que consideramos pueden ser pertinentes.

**SP** Corresponde a un portal interno (intranet) para el control de un proceso productivo sometido un exigente estándar de seguridad. Es el proyecto más

grande y de más duración de la muestra. Hubo diversos requerimientos que se agregaron "sobre la marcha" por lo que la puesta en operación fue postergada en varias ocasiones. El equipo correspondió a tres desarrolladores.

**RE** Es un portal interno para el registro de una serie de casos y documentación asociada, así como la generación de consultas y reportes diversos. Incorpora características de tipo "workflow" (flujo de trabajo) para el seguimiento de los casos registrados. En sus semanas finales la codificación tuvo algunas "interrupciones" debido a solicitudes de pruebas asistidas para los futuros usuarios. El equipo correspondió a $2\frac{1}{2}$ desarrolladores[1].

**AD** Es un portal interno orientado a parametrizar un sistema complejo. Su desarrollo se hizo en forma bastante discontinua (con diversas paralizaciones intermedias.) El equipo correspondió a $1\frac{1}{2}$ desarrolladores.

**AS** Es un portal para uso externo cuyo desarrollo fue muy accidentado y nunca alcanzó un estado "razonable" de calidad; estuvo brevemente en operación pero fue dado de baja por cuestiones comerciales. Puede considerarse como un proyecto incompleto, forzado a operar prematuramente. El equipo correspondió a a $1\frac{1}{2}$ desarrolladores.

**JP** Es una extranet creada con mucha rapidez para satisfacer una necesidad relativamente puntual; interactúa transaccionalmente con un sistema central. El equipo correspondió a a $1\frac{1}{2}$ desarrolladores.

**CT** Es un portal comercial orientado a cotizaciones de productos al por mayor, el cual utiliza y extiende la información de una base de datos ya establecida para un sistema ERP. El equipo correspondió a dos desarrolladores.

**PO** Es un portal de tipo extranet que permite el ingreso y validación de información muy detallada así como su posterior reporte; al igual que el anterior, utiliza y extiende la información de una base de datos ya establecida. El equipo correspondió a dos desarrolladores.

### 3.2.2. Tamaño y Selección de la muestra

La muestra a ser analizada corresponde a siete proyectos ejecutados por el autor; estos proyectos tienen diversa extensión (en artefactos de código fuente constituyentes.)

[1]Las fracciones corresponden a una dedicación a tiempo parcial. Todos estos valores presentan una alta incertidumbre dados los motivos señalados en la sección 1.2.2, por lo que hemos evitado su empleo. Ver también la sección 4.1.2.

| Proyecto | Clave |
|---|---|
| 1 | SP |
| 2 | RE |
| 3 | AD |
| 4 | AS |
| 5 | JP |

Cuadro 3.1: Proyectos de empresa A.

| Proyecto | Clave |
|---|---|
| 6 | CT |
| 7 | PO |

Cuadro 3.2: Proyectos de empresa B.

Para esta selección el autor ha recurrido a la revisión de sus archivos personales, y tras una evaluación del universo total (que como indicamos, consta de aproximadamente 100 proyectos), eligió aquellos que representaban de manera uniforme el perfil tecnológico planteado (software multicapa en lenguaje Java.) En este sentido, no fue necesario recurrir a encuestas, sorteos u otras técnicas de selección.

En particular, sólo se considera el código fuente en lenguaje Java, obviando otros tipos de artefactos[2]. La tabla 3.3 enumera algunos indicadores de la dimensión del código fuente de los productos finales para cada proyecto.

Para el ajuste de modelos no utilizamos la versión actualmente "en operación" de los productos derivados, los cuales pueden haber sido sometidos a modificaciones ulteriores de mantenimiento que están fuera del alcance del proyecto de desarrollo original. Por tal motivo, se considerará como tamaño final de código los valores finales de la muestra que corresponde exclusivamente al proyecto de desarrollo. Estos valores se muestran en la tabla 3.4.

Ver el anexo de Listado de Programas para información acerca de los programas empleados para extraer esta información.

Para la selección de la muestra se han utilizado los siguientes criterios:

1. Se debe disponer del código fuente en su total integridad, y no de estadísticas acumuladas como las que se suelen utilizar en la mayoría de estudios estadísticos de estimación de esfuerzo.

2. Debe considerarse proyectos de desarrollo planificados temporalmente con metas específicas. Esto (lamentablemente) descarta muchos proyectos de

[2]Por ejemplo, elementos gráficos, especificaciones XML de diseño, plantillas JaspeReports, archivos de configuración y metadata, documentación de usuario, etc.

| Proyecto | LOC Total | Palabras | Bytes | Artefactos |
|---|---|---|---|---|
| SP | 76109 | 190124 | 2526083 | 378 |
| RE | 29818 | 76550 | 982222 | 212 |
| AD | 14149 | 32212 | 445156 | 94 |
| AS | 4900 | 14572 | 150899 | 50 |
| JP | 4480 | 10419 | 118961 | 59 |
| CT | 12956 | 36504 | 448336 | 101 |
| PO | 13793 | 34457 | 456602 | 84 |

Cuadro 3.3: Extensión de código fuente de proyectos analizados.

| Proyecto | LOC Muestra |
|---|---|
| SP | 59467 |
| RE | 22258 |
| AD | 7791 |
| AS | 2499 |
| JP | 1685 |
| CT | 9590 |
| PO | 11399 |

Cuadro 3.4: Extensión de código fuente de las muestras finales.

"software libre", donde el esfuerzo es totalmente voluntario y las restricciones temporales prácticamente inexistentes.

3. El proyecto debe haber sido solicitado para satisfacer un requerimiento real no trivial, y preferentemente debe haber entrado en operación. Esto descarta el uso de proyectos simulados (típicamente con equipos de estudiantes) que son comunes en este tipo de investigación.

4. El proyecto tecnológicamente debe corresponder a una arquitectura multicapa. Los componentes del proyecto que no corresponden o son externos a esta arquitectura (en caso de existir), no son considerados como parte de esta investigación.

5. El equipo de desarrollo debe haber estado constituido por un máximo de cuatro personas (en promedio de una a tres), utilizando una metodología altamente adaptada a la forma de trabajo de los clientes que encargan los proyectos.

## 3.3. Técnicas de recolección de Datos

La información actualmente se encuentra físicamente clasificada en forma estructurada a partir de los proyectos originales. Esta clasificación es por proyecto, y por "capa de software", los que se emplazan en sendos directorios o carpetas

de computador. Asimismo, dicha información ya está codificada mediante sus respectivos nombres de archivos y rutas de acceso.

### 3.3.1. Programas de extracción

Como es usual en las organizaciones que producen Software profesionalmente, el código fuente de los proyectos está almacenado en un Sistema de Control de Versiones (SCV), a fin de llevar una historia ordenada y completa de las adiciones, cambios y eliminaciones efectuadas por los analistas. Complementariamente, el SCV puede proporcionar mecanismos de control de acceso a fin de restringir la exposición del código fuente exclusivamente a las personas que lo utilizan.

En nuestro caso, el SCV empleado se llama Subversion (ver Glosario.) En este sistema se proporciona un comando denominado "svn" que permite interactuar con el repositorio de código fuente. Es mediante este comando que se confeccionaron algunos scripts de shell Bash para permitir la extracción de las versiones del código fuente de los proyectos.

En el apéndice de listado de programas se lista un script de shell y un script en lenguaje PHP utilizados para la extracción de los datos.

El resultado es un archivo de texto donde cada línea corresponde a un cambio en un archivo. Estas líneas contienen los campos: nombre de archivo, fecha en formato "timestamp"[3], fecha legible, versión en el repositorio Subversion, analista que registró el cambio en el código, número de líneas agregadas, número de líneas reducidas, y tres datos adicionales obtenidos sobre la última versión del archivo: número de líneas, número de palabras, y número de bytes de cada archivo. A continuación un extracto a modo de ejemplo:

```
AppDAO.java|1265924831|2010-02-11|r3268|U|2|0|12|27|507
AppDAO.java|1263520201|2010-01-14|r3013|U|10|0|12|27|507
XYDAO.java|1265924831|2010-02-11|r3268|U|1|0|20|49|698
XYDAO.java|1264033030|2010-01-20|r3066|U|2|1|20|49|698
```

## 3.4. Procesamiento de datos

### 3.4.1. Exploración Visual

Una de las primeras tareas que se debe realizar como parte del procesamiento de los datos consiste en su exploración visual. A partir de los conjuntos de datos extraídos con el procedimiento antes descrito, es razonable generar histogramas o

[3]Número de segundos transcurridos desde las 00:00 del 1/1/1970; es una codificación típica en sistemas Unix y derivados.

(dado el volumen de la información), gráficos de líneas que describan su evolución temporal de los cambios en el código fuente que han quedado registrados en los "commits" del sistema de control de versiones. Estos cambios son acumulados a distintas escalas como veremos a continuación.

Asimismo, hemos considerado de interés graficar de manera conjunta los siguientes indicadores (acumulados):

1. Número de líneas añadidas
2. Número de líneas eliminadas
3. Número de líneas añadidas más el número de líneas eliminadas
4. Número de líneas añadidas menos el número de líneas eliminadas

El número de líneas añadidas es un indicador que da cuenta del avance "en bruto" del código fuente. Las líneas eliminadas por el contrario están referidas a correcciones o replanteamientos en el desarrollo.

La suma de ambos indicadores da cuenta del trabajo total que se está realizando, pues ambas tareas requieren esfuerzo por parte del desarrollador (más que el esfuerzo del "tipeo", es relevante el análisis que conlleva a una eliminación.)

Finalmente, la diferencia de las líneas añadidas y eliminadas permite dar cuenta del avance "neto" del volumen del código fuente: la integral de esta línea refleja el volumen total de código fuente del producto final.

El gráfico 3.4.1 proporciona una vista del avance de la codificación (del proyecto "RE"[4]) consolidando el número de líneas de código fuente un día a la vez; el alcance corresponde a aproximadamente siete semanas.

En este gráfico cada ordenada corresponde a la suma de líneas de código fuente registradas en el SCV durante el día señalado en el eje horizontal; es decir, la escala de medida del volumen de código fuente es un día. Estrictamente, debería estar conformado exclusivamente por puntos; sin embargo, se ha optado por unir los puntos consecutivos a fin de hacer patente el surgimiento de cualquier patrón de tendencia.

El gráfico muestra un claro patrón semanal, generado de seguro por falta de actividad dominical (para los primeros meses de este proyecto se trabajó los sábados a media jornada.) La leyenda reza: número de líneas añadidas en rojo, eliminadas en amarillo, suma de las anteriores en verde, y la diferencia (avance neto) en marrón.

[4]Este proyecto se ha elegido al azar para este gráfico: todos nuestros proyectos tienen un comportamiento similar.

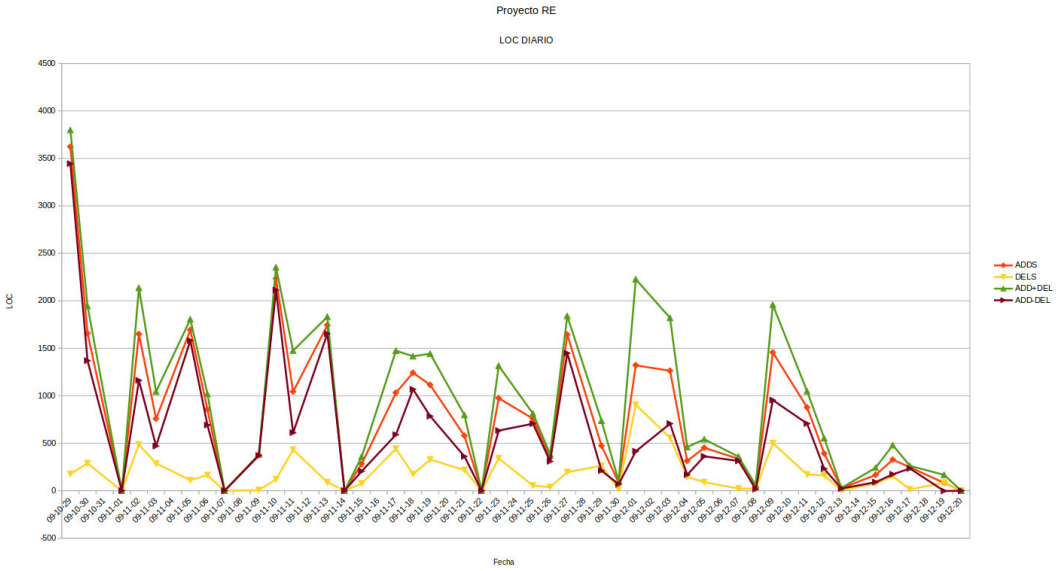



Figura 3.4.1: Codificación a Escala Diaria en proyecto “RE”

Sin embargo, una tendencia más notable se advierte al acumular semanalmente los cambios en el código fuente; evidentemente esto también permite graficar un espacio temporal más amplio, como se muestra en el gráfico 3.4.2.

Este gráfico permite realizar algunas observaciones preliminares. En primer lugar, el patrón general señala una tendencia inicial decreciente, la cual corresponde a una tasa de codificación cada vez menor. Consideramos que este fenómeno se puede explicar por medio de las siguientes ideas que se operan simultáneamente:

- Al principio de cualquier desarrollo, los analistas deben codificar una serie de artefactos (muchas veces incompletos) que sirven como marco general de lo que engloba el proyecto; esto incluye una serie de archivos estandarizados relativamente repetitivos y bastante documentados; en ciertos casos estos pueden ser simplemente copiados de proyectos anteriores.

- Los programadores tienden a desarrollar primero lo que conocen mejor, ya sea por su dominio de la tecnología como por tratarse de los aspectos del dominio de negocio que han sido mejor analizados y/o entendidos.

- Los problemas (subtareas) más sencillos por su propia naturaleza son culminados rápidamente al inicio del proyecto; esto implica que los problemas más difíciles permanecen más tiempo, y su dificultad implica una codificación más pausada para las etapas ulteriores.

- El primer punto del gráfico tiene una interpretación engañosa: es frecuente que el desarrollo se inicie con una serie de prototipos (maquetas) que se emplean para interactuar con el usuario durante el proceso de análisis. En

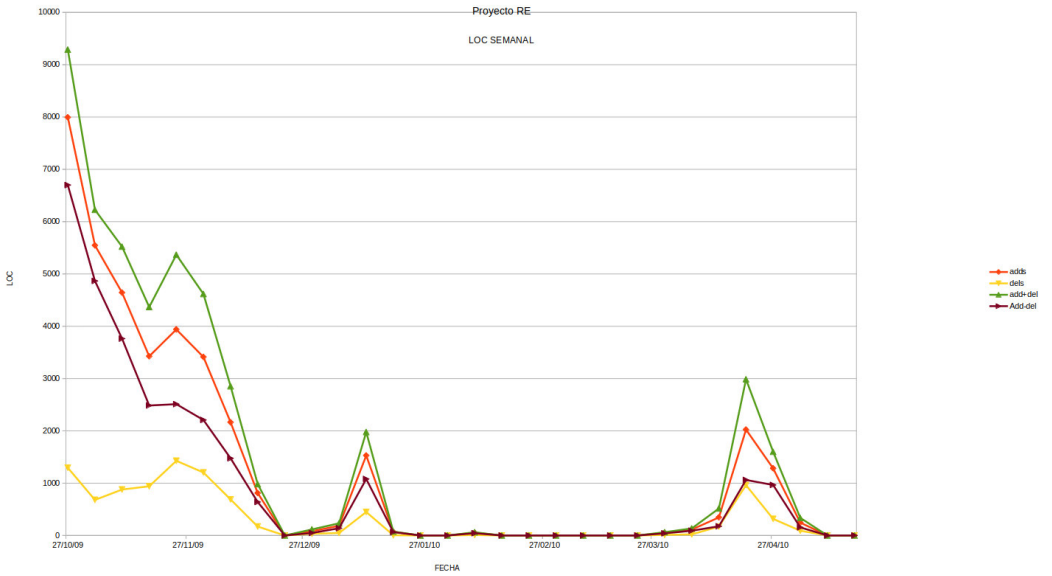



Figura 3.4.2: Codificación a Escala Semanal en proyecto "RE"

ese sentido, también es común que parte de los artefactos del prototipo se reutilicen como punto de partida al momento de inicializar el proyecto en el sistema de control de versiones, lo que implica un pico artificial en los primeros "commits" del código fuente.

Tras este decrecimiento se aprecian dos etapas donde la actividad resurge. A partir de la revisión de las comunicaciones de aquellas fechas, descubrimos que esto es consecuencia de dos revisiones del producto por parte del usuario; de hecho, como se infiere en el gráfico, la primera revisión resultó ser bastante superficial, mientras que la segunda fue más acuciosa, generándose un mayor número de observaciones.

Este patrón ha sido observado en otros proyectos, y resulta lógico hasta cierto punto cuando el usuario está "desconectado" del desarrollo[5].

Como información complementaria, el producto de este proyecto se viene utilizando hasta la actualidad, y ha tenido algunas extensiones y mejoras, las que se deben considerar como "proyectos de mantenimiento". Esto se puede visualizar en el siguiente gráfico donde la acumulación se da a escala mensual; dicho gráfico permite tener una idea global de los "eventos" de mantenimiento realizados a lo largo de varios años[6].

Una forma alternativa de explorar la misma información consiste en un gráfico acumulativo (también conocido como ojiva.) Matemáticamente corresponde a la

[5]Este es uno de los aspectos esenciales de las metodologías ágiles: el usuario debería participar activa y continuamente del desarrollo, con lo cual no existirían estos "picos de actividad" desarticulados del plan de trabajo original. A nuestro entender todo esto es deseable, pero su viabilidad dependerá del contexto laboral de los usuarios.

[6]El pico negativo de abril de 2014 corresponde a una reorganización de algunas clases que compartían métodos muy similares (y voluminosos) en una jerarquía que posibilitó eliminar las duplicaciones.

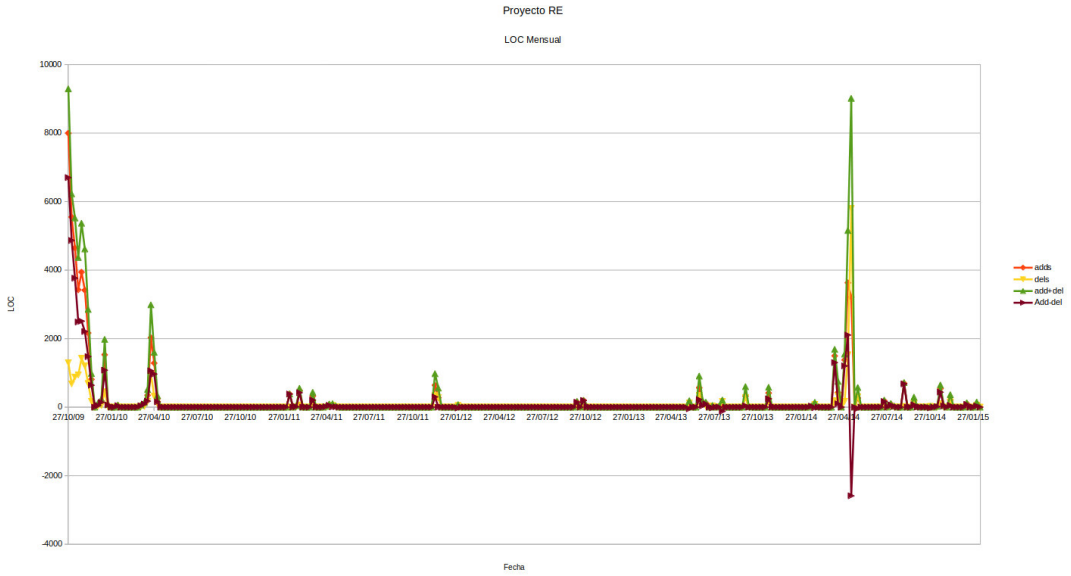



Figura 3.4.3: Codificación a Escala Mensual en proyecto “RE”

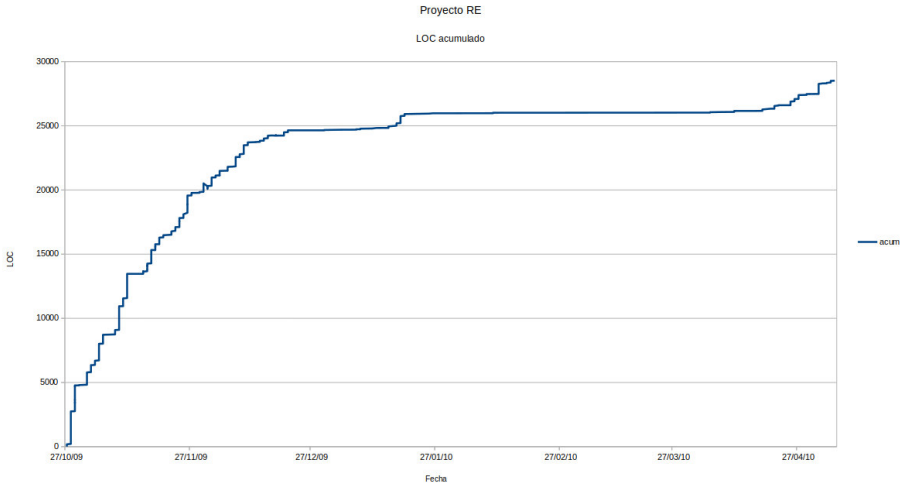



Figura 3.4.4: Gráfico acumulativo de código fuente neto en proyecto “RE”.

integral de cualquiera de los gráficos anteriores (en este caso considerados a escala diaria.) El gráfico 3.4.4 muestra una ojiva para el indicador de código neto (add-del) de las primeras semanas del proyecto “RE”.

Esta forma de gráfica tiene la ventaja de presentar un indicador tangible del producto en construcción en cada instante del tiempo. Analíticamente proporciona una curva que tiende a ser más suave que las de variaciones de cambios, y (salvo situaciones muy excepcionales) tiene una tendencia monótona creciente.

Para complementar esta sección, proporcionamos sendos gráficos de la codificación a escala semanal para el resto de proyectos. No incluimos los correspondientes a escala mensual pues en su mayoría nuestros proyectos tuvieron muy corta duración como para generar un patrón de interés. Asimismo, no incluimos los gráficos acumulativos puesto que visualmente son muy similares al que ya

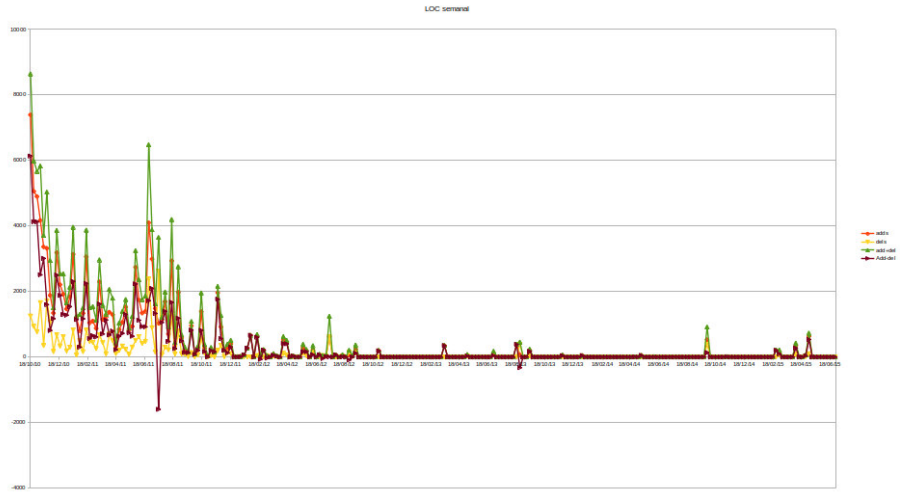



Figura 3.4.5: Codificación a Escala Semanal en proyecto “SP”

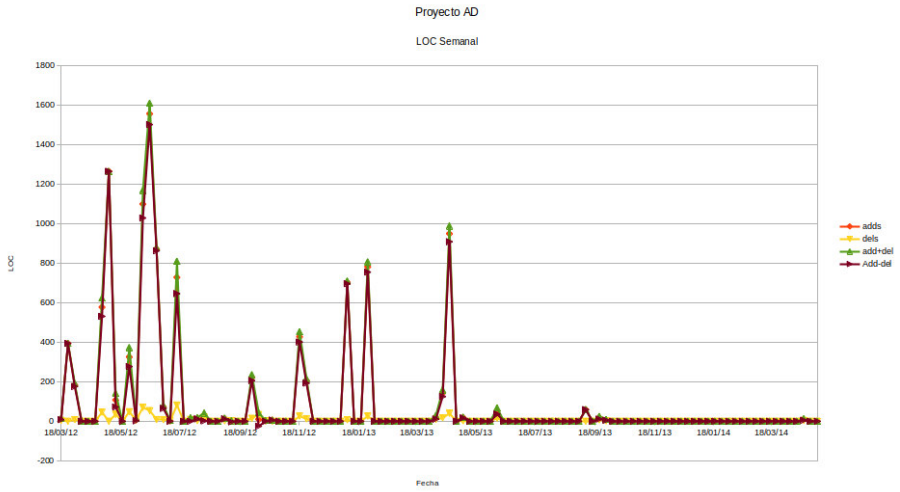



Figura 3.4.6: Codificación a Escala Semanal en proyecto “AD”

presentamos para “RE”.

### 3.4.2. Escalas Temporales

En tanto nuestro interés es la evolución de un proyecto de desarrollo (y no los mantenimientos), optaremos por emplear la **escala semanal** para el modelamiento de las tendencia ideal que modela a escala “macroscópica”: la escala mensual resultaría en una consolidación excesiva con un número de muestras muy reducido. La escala semanal se obtiene sumando las líneas de código fuente agregadas y eliminadas a lo largo de una semana.

De otro lado, la escala diaria proporciona información valiosa localizada a escala “microscópica”, la cual es útil en el modelamiento de la tendencia estocásticas; sin embargo, nuestra experiencia señala que su uso puede introducir distorsio-

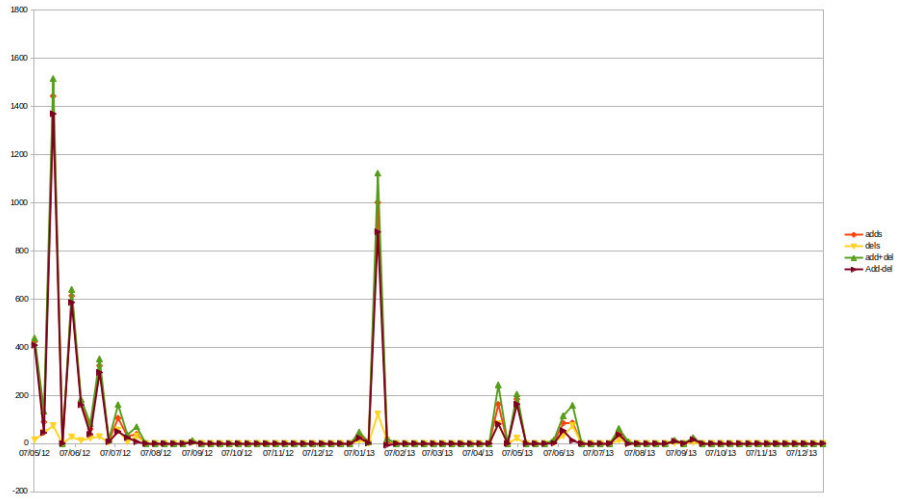



Figura 3.4.7: Codificación a Escala Semanal en proyecto “AS”

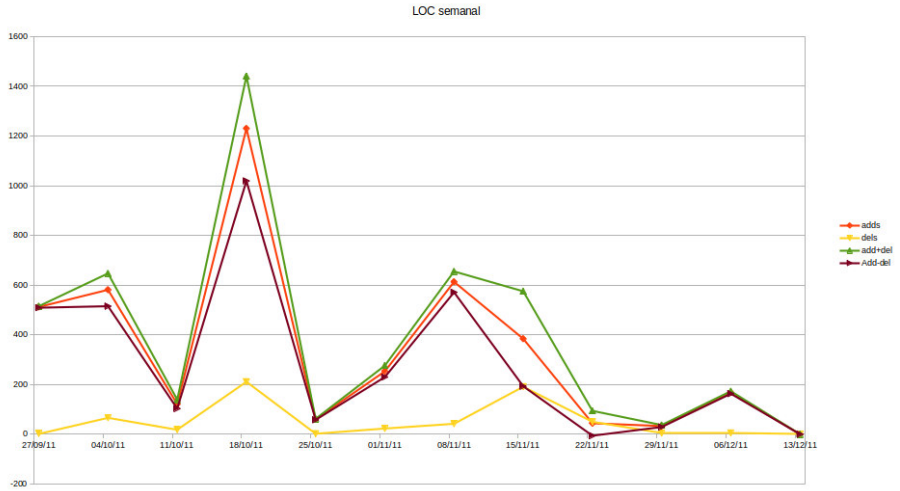



Figura 3.4.8: Codificación a Escala Semanal en proyecto “JP”

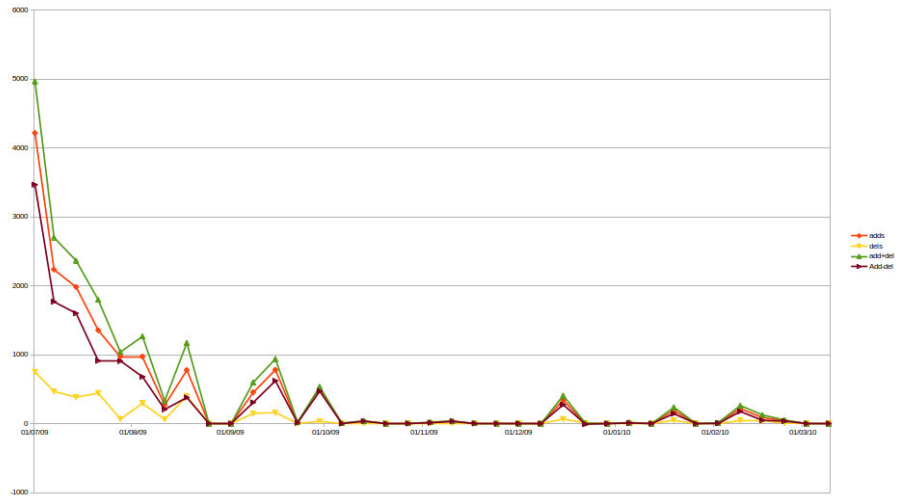



Figura 3.4.9: Codificación a Escala Semanal en proyecto “CT”

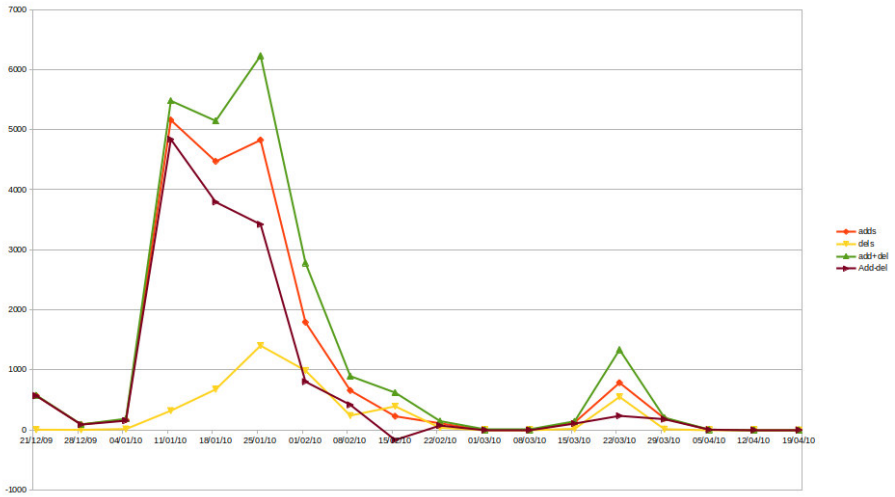



Figura 3.4.10: Codificación a Escala Semanal en proyecto “PO”

nes en las interpretaciones debido a que los desarrolladores de software emplean distintos “ritmos” al realizar sus registros en el SCV; a tal efecto consideramos razonable emplear el **promedio móvil** de tres días[7] .

Tal como vimos en la sección anterior (especialmente en el gráfico 3.4.1), los fines de semana introducen una distorsión en la codificación diaria que no es inherente a la codificación del software, sino a convenciones sociales y regulaciones laborales. Es interesante observar el gráfico en frecuencia de la codificación diaria para el proyecto SP (figura 3.4.11), y el resultado obtenido cuando se eliminan los días correspondientes al fin de semana (figura 3.4.12); en el último caso el pico de siete días ha desaparecido, y en su lugar se aprecian dos picos (mucho menores) correspondientes a cinco y tres días; el primer pico se puede explicar en tanto los participantes del proyecto en forma inconsciente generan “micro-ciclos” de trabajo semanales (en este caso de lunes a viernes) con más frecuencia que con otras duraciones. El segundo pico en realidad es un sobreviviente del primer gráfico, y corresponde al segundo armónico de la frecuencia principal: en tanto la frecuencia principal ahora es $\frac{1}{5}\,días^{-1}$, su segundo armónico corresponde a $\frac{2}{5}\,días^{-1}$ , lo que se refleja en una duración de $5/2 = 2{,}5\ días$ (siendo $7/2 = 3{,}5\ días$ el segundo pico del primer gráfico.) Notar que este análisis de frecuencias se hace más fiable en la medida que el número de muestras es más elevado; en nuestro caso, el patrón observado se aprecia con claridad en los proyectos SP y RE (figura 3.4.13), pero no en el resto de proyectos que contienen un número menor de muestras; para

[7]Esto conlleva la asunción implícita (pero que consideramos razonable a la luz de nuestra experiencia) referida a que los programadores no dejan pasar más de tres días sin registrar sus avances de código fuente en el SCV. Los motivos para esta diversidad de “ritmos” se pueden hallar en las costumbres innatas al desarrollador, así como a la complejidad del artefacto de código fuente sobre el que se está trabajando: cuando la complejidad es elevada es difícil asegurar una razonable corrección del artefacto, lo que retrasa su registro en el SCV.

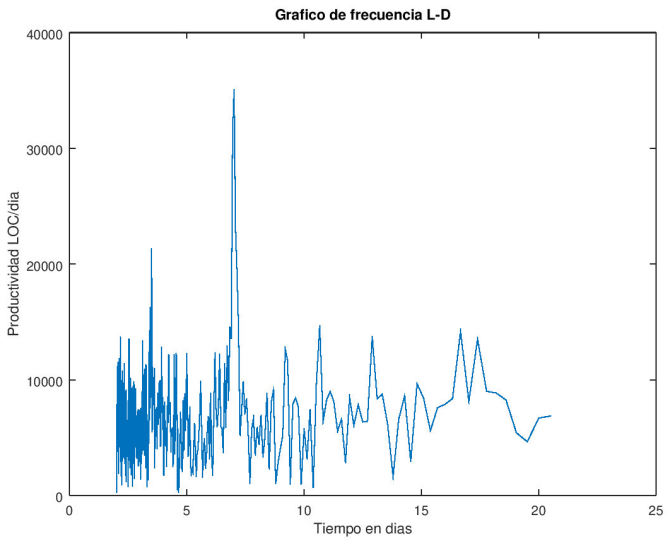



Figura 3.4.11: Gráfico de Frecuencia proyecto "SP" (L-D.)
Considera datos de luenes a domingo. Observar pico de frecuencia de 7 días.

ilustrar esto último la figura 3.4.14 da cuenta del proyecto CT, donde no se percibe ningún patrón (al igual que en el resto de proyectos que no presentaremos por este mismo motivo.)

### 3.4.3. Preparación de los Datos

Nuestro punto de partida es la consolidación a escala diaria (suma de líneas de código añadido y suma de líneas de código eliminado durante un día físico.) Tal como se indicó en la sección anterior, los cambios en la codificación a escala diaria tienen una "distorsión" semanal debido a la ausencia de codificación de los fines de semana, lo que motiva a la eliminación de la actividad correspondiente a sábados y domingo.

Una segunda consideración corresponde la extensión temporal de los datos a ser empleados; al respecto, hemos optado por hacer uso de la información disponible hasta la fecha de puesta en operación del producto terminado, o al menos, de entrega del producto al cliente para su puesta en operación[8]. En ese sentido, es usual que la productividad (como ratio de avance) del software tienda a cero típicamente en las semanas previas al pase a producción cuando el cliente realiza pruebas de certificación, pero también a mitad del desarrollo debido a una multitud de factores externos[9]; tales interrupciones simplemente involucran un "corrimiento temporal" de las muestras disponibles, por lo que nuestra rutina

[8]Esta fecha se obtuvo consultando los registros históricos de las comunicaciones con los clientes.

[9]Estas interrupciones limitan el alcance del modelamiento de la tendencia ideal (semanal) dado su carácter global. Esto complicó las reglas de la sección 4.2.5.2.

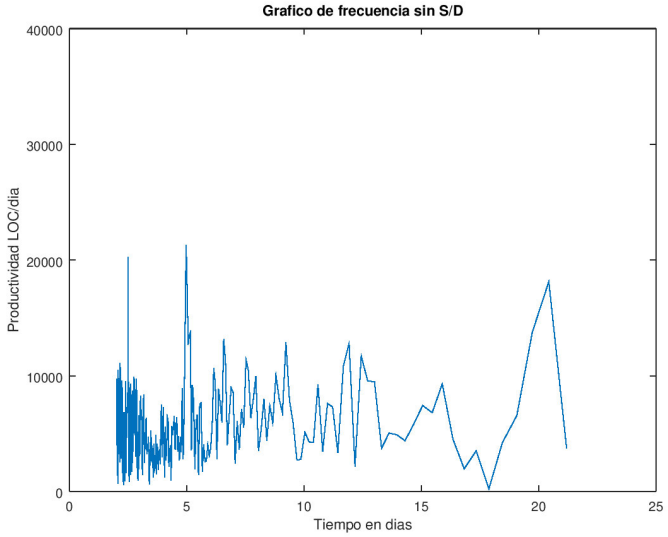



Figura 3.4.12: Gráfico de Frecuencia proyecto “SP” (L-V.)
Considera datos de luenes a viernes. Observar pico (reducido) de frecuencia de 5 días.

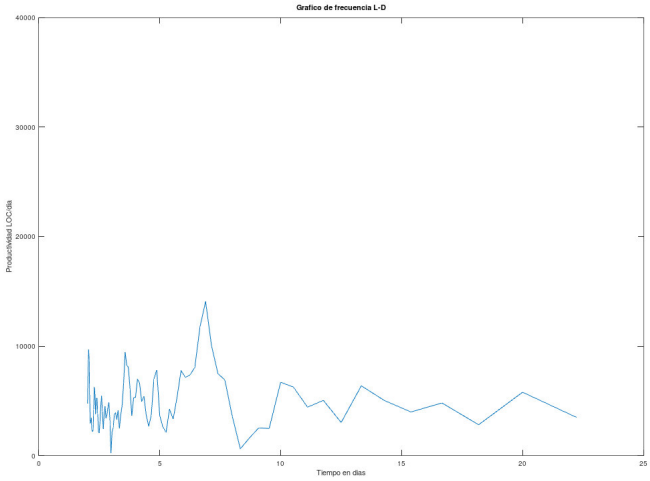



Figura 3.4.13: Gráfico de Frecuencia proyecto “RE” (L-D.)
Considera datos de luenes a domingo. Observar pico de frecuencia de 7 días.

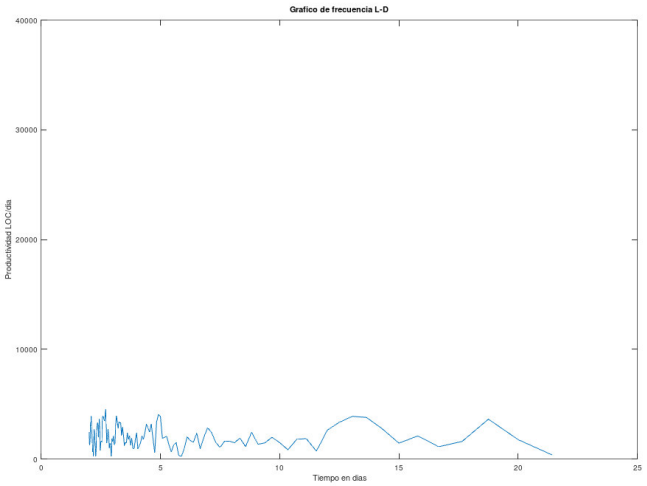



Figura 3.4.14: Gráfico de Frecuencia proyecto “CT” (L-D.)
Considera datos de luenes a domingo. No se percibe el patrón semanal.

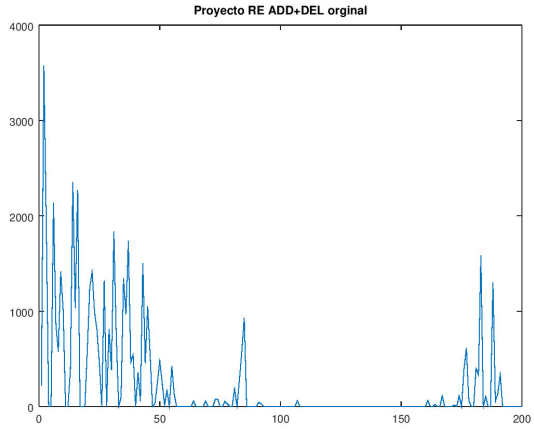



Figura 3.4.15: LOC diario original proyecto “RE”.

de pre-procesamiento de la información realiza este desplazamiento cada vez que transcurren (al menos) cuatro días sin registro de variación en el código fuente. Esto a su vez implica que las tendencias son modeladas sólo durante la ejecución “efectiva” del proyecto. Comparar las figuras 3.4.15 y 3.4.16 correspondientes (respectivamente) a la codificación diaria antes y después de este procesamiento para el proyecto “RE”.

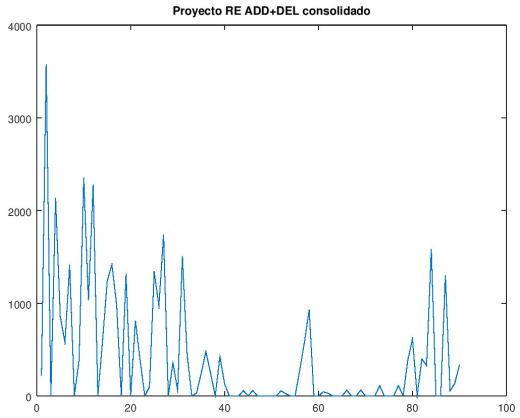



Figura 3.4.16: LOC diario consolidado proyecto “RE”.

# Capítulo 4

# Propuesta de Modelamiento

Este capítulo presenta nuestra propuesta de modelamiento. Las primeras secciones proporcionan una justificación cualitativa de la misma, considerándose los antecedentes y las observaciones discutidas al inicio de este trabajo. Posteriormente se desarrolla un análisis cuantitativo dividido en dos componentes que confluyen en la dinámica del software (se menciona un tercer componente para eventos fortuitos que se excluye del proceso de modelamiento), a los que se denomina tendencia ideal y estocástica, respectivamente.

## 4.1. La Dinámica del Proyecto de Desarrollo de Software

### 4.1.1. El Inicio del Proyecto

No existe una definición estandarizada para lo que normalmente se reporta como inicio del proyecto (Capers Jones, 2008). Sin embargo, esto no nos impide proporcionar algunas nociones que se suelen utilizar de modo informal, pero que idealmente confluyen en un mismo concepto:

a) Declaración de interés: esto corresponde a la puesta en manifiesto de una necesidad, la que se satisface mediante la elaboración de un producto en el marco de un proyecto. Con frecuencia esto deviene en un análisis de factibilidad, consideraciones de presupuesto y aprobación.

b) Aprobación del proyecto: ocurre cuando un interesado que tiene los recursos y el empoderamiento necesario toma la decisión de ejecutar un proyecto de desarrollo. Esta decisión suele implicar la asignación de responsables o la contratación de terceros.

c) Tras lo anterior se suele establecer el inicio formal del proyecto, que es un evento especialmente asociado al ámbito administrativo. En condiciones normales,

esto deviene rápidamente en el inicio del análisis del desarrollo.

d) Inicio del análisis: el personal técnico del proyecto procesa las necesidades manifiestas y las traduce en una especificación que será refinada paulatinamente. Este análisis es considerado normalmente como el verdadero arranque del proyecto desde el punto de vista de los ingenieros y/o analistas.

Dependiendo de la organización en cuestión, es posible que se establezca una etapa adicional de "diseño", que formalmente involucra establecer el "cómo" se implementarán los requerimientos recogidos en el análisis. Es menester señalar que el análisis y el diseño suelen interactuar de modo muy cercano, al punto que en muchos contextos no se hace una distinción formal, prefiriéndose la frase "análisis y diseño" para englobar todo este conjunto de actividades.

Es durante estas actividades de análisis y diseño que se suelen generar los primeros productos de software del proyecto, desplegados en presentaciones o maquetas de apoyo al análisis de cara al cliente, así como en "prototipos de software" que permiten validar una serie de asunciones así como estudiar algunos aspectos tecnológicos en los que el equipo de desarrollo carece de experiencia.

Formalmente, todos los productos de software de esta etapa son estrictamente descartables; en la práctica, es usual que se invierta un esfuerzo significativo en estos productos (especialmente si el software a ser desarrollado es extenso y/o los requerimientos son de difícil comprensión), por lo que se tienden a reutilizar como punto de partida en la etapa de codificación. Asimismo, es usual que el esfuerzo involucrado en estos productos no sea contabilizado como parte de la construcción del software, ya sea por su carácter descartable, como porque suelen desarrollarse de modo relativamente informal conforme los analistas juzgan necesario a partir de las entrevistas con los usuarios finales y/o clientes.

Finalmente, se da inicio a la construcción (codificación) del software, con mucha frecuencia empleando (parcialmente) los productos generados en el análisis y diseño. En este punto es usual que se inicialice el sistema de control de versiones para albergar el código fuente y otros artefactos.

Para nuestro estudio esto último tiene un impacto significativo: en la historia del sistema de control de versiones con frecuencia encontraremos que el software "nace" con cierto contenido desde el primer día. Si estos productos del análisis y diseño son considerables, desde el punto de vista de la evolución temporal es posible que obtener un "pico" artificial al inicio del proyecto, el cual deberá ser anulado para el estudio de la dinámica subsecuente.

De lo anterior se colige que los productos de software se inician mediante dos "fases" sujetas a distintas dinámicas:

1. Como parte del mencionado "análisis y diseño", sujeto a un régimen cuya dinámica viene determinada por criterios comerciales así como por necesida-

des de "comprensión" de la problemática del futuro usuario. Los productos generados están orientados a la visualización del comportamiento de la futura aplicación así como a la interacción humana. El procesamiento de datos está ausente o es simulado. Un extracto de estos productos se emplea como punto de partida del la fase subsiguiente.

2. La etapa de construcción/codificación propiamente dicha, cuya dinámica está determinada por la interpretación detallada de los requerimientos y su traducción a un lenguaje de programación. Dicha traducción no es un asunto trivial, y hasta cierto punto implica completar el diseño de la aplicación a un nivel de operaciones elementales, las que son implementadas mediante las instrucciones y sentencias del ya mencionado lenguaje de programación.

La etapa de construcción/codificación (segunda fase) normalmente tiene una duración mucho mayor que la anterior, además de demandar mayor exclusividad temporal por parte del personal.

Nuestro objetivo está centrado en modelar su dinámica, sin considerar lo correspondiente a la "primera fase". La justificación para esto viene dada por la gran diferencia relativa en el costo de los recursos involucrados[1]; asimismo, desde el punto de vista contractual suele ser mucho más sencillo replantear (o cancelar) el proyecto cuando la codificación propiamente dicha no se ha iniciado.

### 4.1.2. Los recursos humanos asignados al proyecto

Las observaciones de Norden empleadas en el modelo de Putnam anteriormente explicado, tienen validez experimental para grandes proyectos de diversas industrias, y desde el punto de vista cualitativo son compatibles con el sentido común; a saber, al inicio del proyecto el número de participantes se incrementa conforme pasan los días (lo que se puede adjudicar a la gradual diversificación de actividades, a la contratación de especialistas, al gradual despliegue de la logística necesaria para que los trabajadores realicen las actividades, etc.) Este "crecimiento" eventualmente alcanza un pico donde la fuerza de trabajo se estabiliza, para posteriormente iniciar su declive conforme las actividades se completan.

Sin embargo, esta "dinámica de asignación de recursos" es cuestionada para el caso de los proyectos de software (ver por ejemplo K. Pillai, Nair y S (1997)) específicamente en cuanto al periodo de crecimiento de la fuerza de trabajo: en proyectos medianos (y con más razón en los pequeños) la tasa de crecimiento es superior a la prescrita por la ecuación de Norden, lo que justifica propuestas alternativas como las ya señaladas.

[1]El costo del análisis y diseño suele ser significativo; sin embargo, aquí nos referimos únicamente al costo de los productos de software generados en tal etapa.

En nuestro estudio no consideramos el problema de la asignación de los recursos humanos en razón a que:

1. Nuestro análisis de la dinámica está basado en los productos de código fuente el cual sólo se produce cuando el proyecto ya está bien avanzado (específicamente, en la etapa de codificación siguiendo la terminología tradicional.)

2. En nuestros casos de estudio los equipos de desarrollado han sido pequeños (de dos a cinco programadores) y se han mantenido relativamente inalterados desde que el arranque de la codificación; estos desarrolladores asimismo se han hecho cargo de las pruebas, correcciones y soporte del producto.

### 4.1.3. Modelamiento Cuantitativo

El modelamiento cuantitativo intenta abstraer el comportamiento de los desarrolladores expresado en el conteo del código fuente de los proyectos de software. Este modelamiento se implementa mediante la distinción de un conjunto de "tendencias" que consideramos subyacen al proceso de desarrollo del proyecto. Estas tendencias son empleadas para generar los modelos cuantitativos según su naturaleza.

En los siguientes párrafos utilizamos la siguiente notación que aquí presentamos en forma resumida:

$N(t)$ el número total de líneas de código fuente en el proyecto en el instante $t$ (según la unidad de medida temporal elegida.)

$N_i = N(t_i)$ es el valor para $N(t)$ para el momento $t = t_i$. Para $t = 0$ utilizamos abreviadamente la notación $N_0 = N(0)$; asimismo, para el momento correspondiente al fin del proyecto denotado por $t = t_F$, utilizamos la notación abreviada $N_F = N(t_F)$.

$\widehat{N_i} = N_{mod}(t_i)$ es el valor que estimado por el modelo "*mod*" (predicción) para $N(t)$ en el instante $t_i$. Por ejemplo, en el contexto del modelo $ELI$ se tiene $\widehat{N_i} = N_{ELI}(t_i)$.

$N_I(t)$ es función que aproxima $N(t)$ en el contexto del modelamiento que denominamos "tendencia ideal".

$N_E(t)$ es función que aproxima $N(t)$ en el contexto del modelamiento que denominamos "tendencia estocástica procesual".

$D(N)$ es una función que modela la "dificultad" del desarrollo en relación a la cantidad de líneas de código fuente ($N$ unidades) en razón a su complejidad asociada.

**a) Tendencia Ideal**

Corresponde a la implementación de los requerimientos especificados en la etapa de análisis y diseño. Esta implementación está a cargo de un equipo cuya "fuerza de trabajo" permanecerá constante hasta el término del proyecto. Se asume que siendo desarrolladores competentes, el avance de su codificación sólo dependerá de la "dificultad" de entendimiento del problema y su traducción al lenguaje de programación correspondiente, lo cual a fin de cuentas dependerá de su aptitud mental y experiencia profesional.

En particular nos interesa modelar el avance del código fuente total (líneas de código) generado por el equipo de desarrollo a lo largo del tiempo, para lo cual definimos la función $N_I$:

$$\begin{aligned} N_I : [0, t_F] \subset \mathbb{R} &\to \mathbb{R} \\ t &\mapsto N_I(t) \end{aligned}$$

donde $N_I(t)$ corresponde al valor estimado por el modelo para la cantidad de líneas de código fuente en el proyecto en el instante $t$.

En tanto esta función es de carácter "acumulativo", le es inherente un efecto de "suavizado" que puede ocultar información valiosa. Este problema se refleja en los estudios de asignación de recursos a través del tiempo como los que se discutió anteriormente:

El gráfico acumulativo suaviza todas las características interesantes. Es muy difícil predecir algo a partir de este tipo de gráfico. (Warburton, 1983, p.564)

Es por esto que optamos por emplear la función alternativa de productividad del equipo de desarrollo, considerada como "razón de cambio" del avance de las líneas de código fuente:

$$Productividad = N'(t) = \frac{dN(t)}{dt}$$

Esta función viene dada en líneas de código por unidad de tiempo.

Señalemos que esta tendencia es denominada "ideal" en la medida que un proyecto de software normalmente se da en un contexto sistémico mayor al que comprende el equipo de desarrollo; asimismo, el equipo de desarrollo no está determinado únicamente por su nivel de competencia por lo que su productividad no es una función fácilmente parametrizable en el largo plazo[2].

[2]Por ejemplo, muchos proyectos de software son en realidad un conjunto de actividades constituyentes de un proyecto mayor (por ejemplo, de tipo comercial.) Asimismo, ya nos hemos referido a los múltiples factores que determinan variaciones en la productividad y los estimados de esfuerzo que son ampliamente documentados para modelos como COCOMO y similares.

**b) Tendencia Estocástica Procesual**

Comprende los principales eventos no considerados en la tendencia ideal, los cuales generan incertidumbre en el devenir del proyecto. Más concretamente, nos referimos a todo aquello que motive codificación adicional a la especificación, con el consiguiente incremento en el esfuerzo total así como el retraso del tiempo de finalización del proyecto.

Al respecto nuestro planteamiento reduce estos eventos a:

- Cambios/extensiones en los requerimientos por parte del usuario/cliente del futuro programa
- Errores e inconvenientes diversos

En este caso el modelamiento corresponde a una función $N_E$ que describe el número total (acumulados) de los mencionados eventos para un día $t$, lo que corresponde a un proceso de conteo (sección 2.2.2):

$$\begin{aligned} N_E : [0, t_F] \subset \mathbb{Z} &\to \mathbb{N} \cup \{0\} \\ t &\mapsto N_E(t) \end{aligned}$$

Como veremos en la sección 4.3.1, parte de nuestra propuesta de modelamiento consiste en aplicar los modelos de tendencia ideal para el valor esperado del proceso estocástico:

$$E[N_E(t)] \approx \lfloor h(N_I(t)) \rfloor \tag{4.1.1}$$

Aquí $E[.]$ denota el valor estimado para una variable aleatoria, y $h$ es alguna función de variable real monótonamente creciente lo que implica proporcionalidad directa.

En lo que sigue, se empleará el símbolo $N$ para denotar tanto a $N_I$ como $N_E$ según el contexto de discusión (el subíndice se reserva para hacer referencia a la posición temporal.) Cuando se desea distinguir el valor estimado por el modelo frente a la muestra correspondiente, el primero se denotará mediante $\widehat{N_i}$, reservándose $N_i$ para la segunda.

**c) Tendencia Estocástica de Fondo**

Corresponde a todos aquellos eventos que distorsionan la tendencia ideal pero que no pueden ser adjudicados a la tendencia estocástica procesual. Aquí se tiene, por ejemplo, problemas climatológicos, enfermedades y molestias de los participantes, desacuerdos contractuales, siniestros diversos, etc.

En cualquier actividad humana esta tendencia está presente y sólo puede ser parcialmente mitigada con las medidas de prevención correspondientes. En ese sentido, no participará del modelo.

**Nota:** La combinación de tendencias de diversas características se emplea con frecuencia en la economía, donde es de interés para el estudio de los patrones repetitivos en los indicadores del mercado (ciclos) así como los de naturaleza disruptiva:

"Las innumerables tendencias que provocan desequilibrio económico pueden clasificarse de manera general en tres grupos: a) tendencias de crecimiento o trayectoria, que son estables; b) perturbaciones fortuitas, que son inestables, y c) tendencias cíclicas, que son inestables pero que se repiten consistentemente." (Fisher, 1999, p.191)[3] En nuesto estudio no se consideran las tendencias cíclicas; sin embargo, anotemos que éstas sí intervienen en los análisis de la evolución del software a largo plazo[4].

## 4.2. Modelamiento de la Tendencia Ideal

Nuestra hipótesis de trabajo es:

> La Dinámica de Proyectos de Desarrollo de Software Multicapa evoluciona cuantitativamente a partir del Código Fuente.

A partir de la hipótesis general, planteamos la siguiente hipótesis específica:

> En la etapa de codificación, la dificultad en la construcción del software tiende a crecer debido al mayor número de interrelaciones que existe entre el software que se escribe y el software ya escrito. Esta dificultad no crece ilimitadamente; por el contrario, alcanza un régimen estable.

Sea $N(t)$ la cantidad de código fuente de la que dispone el proyecto para el día $t$, y sea $D = D(N)$ una función que refleja la dificultad del desarrollo cuando la cantidad del código fuente es igual a $N$ unidades.

Esta "dificultad" se refiere a la resistencia que percibe el desarrollador al intentar agregar líneas de código fuente debido a la complejidad generada por las $N$

[3]La cita pertenece a la introducción de un artículo acerca de la teoría económica denominada "deuda-deflación" postulada por Irving Fisher para explicar el "crash" de la bolsa de 1929, y la posterior depresión.

[4]Cfr. con la literatura asociada a las "leyes de evolución del software", a las que se hizo referencia en la sección 1.2.1.

líneas de código ya existentes. Al respecto, definimos $D(N)$ mediante la ecuación diferencial:

$$\frac{dN}{dt} = \frac{k}{D(N)} \tag{4.2.1}$$

donde $k > 0$ es una constante que depende de las unidades en que se expresan las funciones participantes. La función $D(N)$ está sujeta a las consideraciones de la hipótesis, lo que se reflejará en condiciones para la última ecuación diferencial como se verá en la siguiente sección.

La dificultad o resistencia que presenta el trabajo de desarrollo opera a dos escalas: una local y otra global. La escala local comprende la representación mental que debe alcanzar el programador de las instrucciones y sentencias de control que proporciona el lenguaje, lo que ha sido estudiado desde un punto de vista lexicográfico y estructural (ejemplificados en nuestros antecedentes mediante la "ciencia del software" y la complejidad de McCabe (Halstead y col., 1977; McCabe, 1976)), así como del proceso mental a corto plazo del desarrollador (Wang, 2003). Por el contrario, la escala global está referida a las tácticas planteadas y ejecutadas por el programador[5] a fin de agregar o modificar alguna funcionalidad de la aplicación; lo que se consigue mediante la intervención y/o invocación a diversos componentes o módulos. Esto involucra una actitud más reflexiva (en comparación con la escala local, más analítica) a la vez que defensiva, pues es altamente probable introducir fallos dispersamente localizados y más difíciles de detectar. La escala global análoga a la "complejidad espacial" (Chhabra, Aggarwal y Singh, 2003), y como tal es impactada por el tamaño del código fuente (directamente proporcional.)

Salvo en dominios muy especializados, en cualquier aplicación no trivial la dificultad a escala global eventualmente domina a la de escala local. Esta última se constituye en un componente menor, salvo en algunas rutinas particularmente críticas o sensibles[6].

### 4.2.1. Planteamiento Analítico de Tendencia Ideal

Siguiendo con las hipótesis, la dificultad se incrementa conforme aumenta el tamaño del código; esto es, $D'(N) > 0$; sin embargo, este incremento no es ilimi-

[5]Como complemento al nivel táctico, las actividades de diseño corresponden al nivel "estratégico" que las determina.

[6]La descomposición de la complejidad a distintas escalas no es una idea nueva. Por ejemplo en Harrison y Cook (1987) se propone la métrica de complejidad "Micro/Macro" (MMC) calculada como la sumatoria de los productos de las complejidades "micro" y "macro" de los subprogramas constituyentes, donde éstas últimas desempeñan un papel análogo al de nuestras escalas local y global, respectivamente. En nuestra opinión el uso de la operación de multiplicación tiende a sobrerepresentar el efecto de la complejidad "micro" en proyectos de envergadura no trivial.

tado, sino que eventualmente recede. Esto se puede plantear de diversas formas, entre las cuales quizá la más sencilla es considerar la diferencia entre una cota superior igual a 1 para $D(N)$ y el valor de esta función $1 - D(N)$, la cual debe anular la tasa de crecimiento de $D$ (es decir, $D'$) conforme esta magnitud se acerca a $b$:

$$D'(N) = a(1 - D(N))$$

Incluso al inicio (con $N = N_0$) existe cierta dificultad inicial para la codificación; luego $D(N_0) = D_0 > 0$, con lo que tenemos un problema de Cauchy de integración inmediata:

$$D(N) = 1 - \exp(-aN + C_1)$$

Introduciendo la condición inicial:

$$D_0 = 1 - \exp(-aN_0 + C_1) \Rightarrow C_1 = aN_0 + ln(1 - D_0) \rightarrow$$

$$D(N) = 1 - (1 - D_0)\exp(-a[N - N_0])$$

De aquí es claro que $D(N_0) = D_0$ como era de esperarse.

Al analizar el rango de esta función, es claro de $\lim_{N\to\infty} D(N) = 1$ se tiene que su variación corresponde al intervalo $[D_0, 1)$.

De otro lado, dado 4.2.1 ahora tenemos:

$$\frac{dN}{dt} = \frac{k}{1 - (1 - D_0)\exp(-a[N - N_0])} \tag{4.2.2}$$

Esta ecuación diferencial puede integrarse por el método de separación de variables; sin embargo, el resultado no arroja una expresión explícita para $N(t)$ que sería lo deseable:

$$\int [1 - (1 - D_0)\exp(-a[N - N_0])]dN = \int k.dt \Rightarrow$$

$$N + \frac{(1 - D_0)\exp(-a[N - N_0])}{a} = kt + C$$

Usando el valor inicial $N(0) = N_0$:

$$C = N_0 + \frac{1 - D_0}{a} \Rightarrow$$

$$(N - N_0) + \frac{(1 - D_0)[\exp(-a[N - N_0]) - 1]}{a} = kt \tag{4.2.3}$$

Esta ecuación determina a $t$ en términos de $N$ (en esta sección denominamos $T(N)$ a la regla de correspondencia):

$$t = T(N) = \frac{1}{k}\{(N - N_0) + \frac{(1 - D_0)[\exp(-a[N - N_0]) - 1]}{a}\} \tag{4.2.4}$$

Donde $T : [N_0, \infty) \to [0, \infty)$. Sin embargo, en la práctica estamos interesados en saber cómo evoluciona $N$ a través del tiempo: $N(t)$, lo que constituye una función inversa a $T(N)$:

$$\begin{aligned} T^{-1} : [0, \infty) &\to [N_0, \infty) \\ t &\mapsto N(t) \end{aligned}$$

Para la existencia de tal inversa es necesario y suficiente que $T(N)$ sea biyectiva:

En particular, si $f : D \to E$ es uno a uno, entonces para cada $y \in f(D)$, existe un único $x \in D$ tal que $f(x) = y$. En este caso, escribimos $x = f^{-1}(y)$. Así obtenemos una función $f^{-1} : f(D) \to D$ tal que $f^{-1}of = id_D$ y $fof^{-1} = id_{f(D)}$. (Ghorpade y Limaye, 2008, p.16)

Asimismo, si la función es estrictamente creciente en su dominio, entonces es inyectiva en el mismo:

Sea $I$ un intervalo conteniendo más de un punto, y $f : I \to \mathbb{R}$ una función diferenciable. Entonces tenemos lo siguiente: [...] $f'$ es positiva a lo largo de $I \Rightarrow f$ es estrictamente creciente en $I$. (Ghorpade y Limaye, 2008, p.125)

Dado que $T(N)$ viene expresada mediante funciones y operaciones elementales, es diferenciable en todo $\mathbb{R}$. Luego, es suficiente verificar que esta función es estrictamente creciente. En efecto, derivando 4.2.4:

$$T'(N) = \frac{1}{k}\{D_0 \exp(-a[N - N_0])\}$$

Siendo la función exponencial positiva en los reales, como $D_0$ (dificultad inicial positiva) y $k$ (constante de proporcionalidad asumida positiva):

$$T'(N) > 0$$

Luego $T$ es inyectiva y existe la función $N(t) = T^{-1}(t)$.

La función $N(t)$ definida de esta manera será denominada "ELI" (exponencial-lineal-inversa), y es nuestra primera propuesta de modelamiento para la "tendencia ideal".

### 4.2.2. Caracterización de la función ELI

Volviendo a la ecuación diferencial (4.2.2), para $t = 0$ se tiene que $N = N_0$ por tanto:

$$\frac{dN}{dt}|_{t=0} = \frac{k}{D_0} \tag{4.2.5}$$

esto corresponde a la razón de cambio del tamaño del código fuente al principio del proceso; es decir, la productividad inicial. Este es un dato que se puede obtener empíricamente al inicio del proyecto.

El denominador del segundo miembro de 4.2.2 es estrictamente positivo pues $N \geq N_0 \Rightarrow -a(N - N_0) \leq 0 \Rightarrow exp(-a(N - N_0)) < 1 \Rightarrow (1 - D_0)exp(-a(N - N_0)) < (1 - D_0) \Rightarrow -(1 - D_0)exp(-a(N - N_0)) > -(1 - D_0) \Rightarrow 1 - (1 - D_0)exp(-a(N - N_0)) > D_0$, por lo cual el denominador es positivo, entonces:

$$\frac{dN}{dt} > 0 \tag{4.2.6}$$

lo que significa que el código fuente (idealmente) tiene una tendencia monótonamente creciente. Debido a esto, la tendencia de la productividad conforme el código crece (ilimitadamente) se obtiene mediante:

$$\lim_{N \to \infty} \frac{dN}{dt} = k \tag{4.2.7}$$

Todos los términos de esta expresión son positivos. Luego:

**Proposición.** *Cuando el tamaño del código fuente crece ilimitadamente (lo que puede considerarse una primera aproximación al estado final del proyecto), la productividad final tiende a estabilizarse en una tasa constante.*

Otra observación está referida a la evolución de la productividad, para lo que observaremos la derivada de 4.2.2 (abreviaremos el denominador con el símbolo $\Delta$):

$$\frac{d^2N}{dt^2} = -a\frac{k}{\Delta^2}\exp(-a[N - N_0])\frac{dN}{dt} \Rightarrow$$

$$\frac{d^2N}{dt^2} = -a\frac{k}{\Delta^2}\exp(-a[N - N_0])\frac{k}{\Delta} = -\frac{ak^2\exp(-a[N - N_0])}{\Delta^3} \tag{4.2.8}$$

Como el denominador $\Delta$ es estrictamente positivo, resulta que:

$$\frac{d^2N}{dt^2} < 0 \tag{4.2.9}$$

y se tiene el siguiente resultado, que es consecuente con nuestra asunción de "dificultad creciente":

**Proposición.** *La función de productividad ideal de la codificación es estrictamente decreciente.*

La asíntota señalada en la ecuación 4.2.7 así como la concavidad de 4.2.9 y el valor inicial $(t_0, N_0)$ permite esbozar la curva $N(t)$ para este modelo. La figura 4.2.1 permite apreciar un ejemplo (línea discontinua.)

Siendo la función estrictamente decreciente con un valor límite dado por 4.2.7 entonces se tiene:

$$\frac{dN}{dt} \geq k \tag{4.2.10}$$

El tamaño del código fuente para el instante $t$ se obtiene mediante mediante la acumulación del avance de la codificación desde el valor inicial $N_0$:

$$N(t) = N_0 + \int_0^t \frac{dN}{dt} dt$$

Debido a 4.2.10, usando el teorema 6.12-b de Rudin y col. (1976):

$$\int_0^t \frac{dN}{dt} dt \geq \int_0^t k dt = kt$$

Luego, conforme $t \to \infty$ , el valor de la función de tamaño del código fuente:

$$N_0 + \int_0^t \frac{dN}{dt} dt$$

se hace infinito. Sin embargo, puesto que el problema que resuelve el software es finito, mientras la fuerza de trabajo se mantiene operativa, entonces:

**Corolario.** *El proyecto necesariamente termina en un tiempo finito.*

#### 4.2.2.1. Aproximación Heurística de la función ELI

Una aproximación inicial tentativa al tamaño total del código fuente $N_F$ consiste en asumir que la curva productividad declina a un ratio constante (segunda derivada de $N$) desde su valor inicial, lo cual corresponde a un escenario idealizado en el cual la dificultad se incrementa hasta el infinito (productividad cero) deteniendo el desarrollo por completo. En tal caso empleando 4.2.8, se tiene:

$$N' \approx \frac{k}{D_0} + t(\frac{d^2N}{dt^2}|_{t=0}) = \frac{k}{D_0} - t\frac{ak^2}{D_0^3}$$

Esta función alcanza el valor cero en el instante $t_p$ (donde denotamos $N(t_p) = N_p$):

$$\frac{k}{D_0} = t_p \frac{ak^2}{D_0^3} \Rightarrow t_p = \frac{D_0^2}{ak}$$

En tal caso el área bajo la curva de $N'$ (el tamaño del código fuente) en el intervalo $[0, t_p]$ corresponde a un triángulo (usando (4.2.5)):

$$N_p = \frac{N'(0)t_p}{2} = \frac{(\frac{k}{D_0})(\frac{D_0^2}{ak})}{2} = \frac{D_0}{2a}$$

En un proyecto de mediana envergadura podemos esperar que el tamaño total del código fuente sea varias veces esta medida. Si requerimos hacer una estimación iterativa de los parámetros de estas ecuaciones, un punto de partida razonable es por tanto:

$$N_{Total} \approx \frac{D_0}{2a} \tag{4.2.11}$$

donde $\alpha$ es un coeficiente mayor que la 1/2 (pues se ha incorporado el factor dos del denominador de la expresión anterior), con el cual podemos obtener diversas soluciones.

De 4.2.3 considerando que $N \gg N_0$ y que $\exp(-a[N - N_0]) \to 1$:

$$N_F = kt_F$$

Usando la aproximación $N_F \approx N_{Total}$:

$$\frac{D_0}{2a} = kt_F \Rightarrow a = \frac{D_0}{2kt_F} = \frac{1}{2(\frac{k}{D_0})t_F} \tag{4.2.12}$$

Reemplazando en 4.2.11 se encuentra $D_0$:

$$D_0 = N_F(2a) = \frac{N_F}{(\frac{k}{D_0})t_F} \tag{4.2.13}$$

Con este valor, se obtiene $k$ puesto que como señalamos $\frac{k}{D_0}$ puede obtenerse de los datos experimentales:

$$k = (\frac{k}{D_0})D_0 \tag{4.2.14}$$

**Ejemplo.** A continuación ilustramos el uso de la aproximación anterior para las primeras semanas del proyecto CT; notar que este modelo fue elegido al azar, y siendo un ejercicio numérico auxiliar no se repetirá con el resto de proyectos por ser redundante.

- Proyecto: CT
- $T_F = 15$ semanas (muestra filtrada según criterios de la sección 3.4.3)

- $N_F = 9590$
- $N_1 = 3383$
- $N_0 = 1609$

La primera semana la productividad inicial fue: $\frac{k}{D_0} = (3383 - 1609) = 1774$ líneas/semana

Aplicando 4.2.12, 4.2.13 y 4.2.14:

$$a = \frac{1}{2(\frac{k}{D_0})t_F} = 0{,}0002086$$

$$D_0 = \frac{N_F}{(\frac{k}{D_0})t_F} = 0{,}1391$$

$$k = (\frac{k}{D_0})D_0 = 1774 * 0{,}1391 = 246{,}82$$

Reemplazando en 4.2.2 nuestro modelo inicial resulta ser:

$$\frac{k}{1 - (1 - D_0)\exp(-a[N - N_0])}$$

$$\frac{dN}{dt} = \frac{246{,}82}{1 - (1 - 0{,}1391)\exp(-0{,}0002086[N - 3383])}$$

Obteniendo una solución computacional vía el algoritmo de Runge-Kutta (Burden, Faires y D, 2010) con 6 pasos (ver listado de código fuente en anexos) se tiene el resultado mostrado en la tabla 4.1, el mismo que se grafica en la figura 4.2.1. El error final para $\widehat{N_F}$ ha sido de casi 7 %, lo que se puede mejorar iterativamente incrementando el parámetro $\alpha$ de este método; sin embargo, ese no es nuestro objetivo: la aproximación heurística sólo será empleada para encontrar un conjunto de parámetros iniciales del modelo ELI, el cual será posteriormente ajustado mediante la optimización de una función de error total.

#### 4.2.2.2. Ajuste de ELI mediante Optimización

En la práctica es usual disponer de un conjunto de muestras $(t_i, N_i)$ de un proyecto real; a fin de analizar la bondad del modelo, debemos estudiar la forma de ajustarlo a estos datos. A tal efecto, usando (4.2.4) para cada valor $t_i$ se puede obtener (aproximadamente) $N_{ELI}(t_i) = \widehat{N_i}$ resolviendo la ecuación mediante métodos numéricos:

| Semana | $\widehat{N_i}$ | $N_i$ | $\widehat{N_i} - N_i$ | $err\,\%$ |
|---|---|---|---|---|
| 0 | 3383.00 | 3383 | 0.00 | 0.00 % |
| 1 | 4539.75 | 4992 | -452.24 | -9.05 |
| 2 | 5251.77 | 5824 | -572.22 | -9.82 |
| 3 | 5837.87 | 7084 | -1246.12 | -17.59 |
| 4 | 6355.45 | 7745 | -1389.54 | -17.94 |
| 5 | 6828.55 | 8193 | -1364.44 | -16.65 |
| 6 | 7269.95 | 8727 | -1457.04 | -16.69 |
| 8 | 7687.42 | 8752 | -1064.57 | -12.16 |
| 9 | 8086.06 | 9068 | -981.93 | -10.82 |
| 10 | 8469.46 | 9114 | -644.53 | -7.07 |
| 11 | 8840.24 | 9117 | -276.75 | -3.03 |
| 12 | 9200.39 | 9214 | -13.60 | -0.14 |
| 13 | 9551.47 | 9378 | 173.47 | 1.84 |
| 14 | 9894.68 | 9590 | 304.68 | 3.17 |
| 15 | 10231.05 | 9590 | 641.05 | 6.68 |

Cuadro 4.1: Resultados de ajuste heurístico para CT.

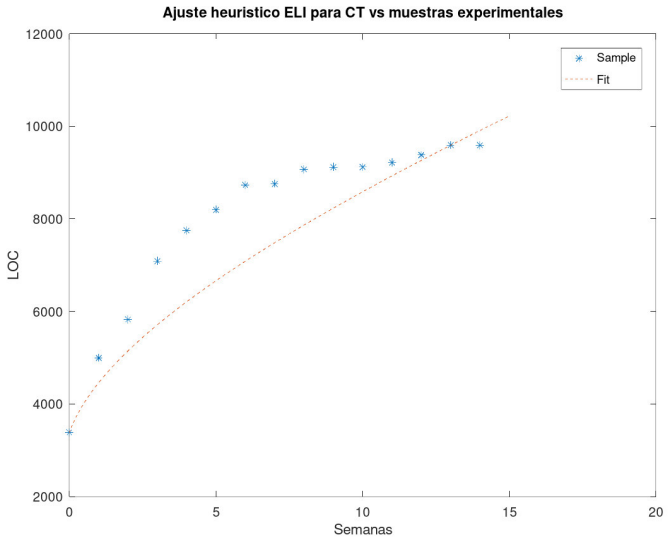



Figura 4.2.1: Modelo ELI para Tendencia Ideal $N(t)$.
Versus muestras experimentales.

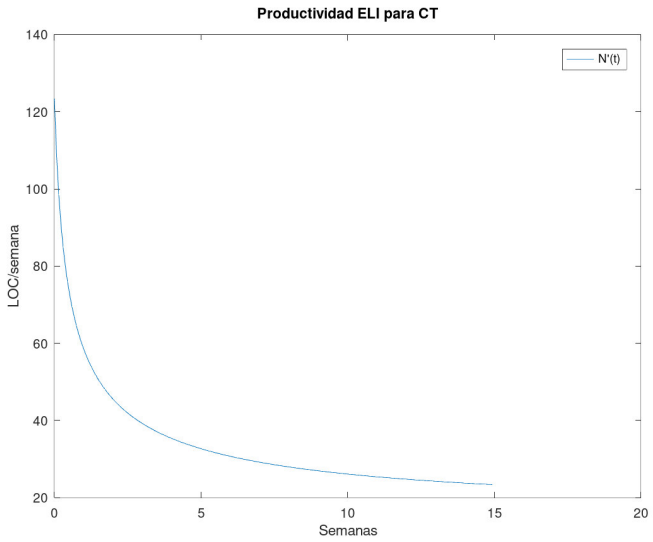



Figura 4.2.2: Modelo ELI para Productividad Ideal $N'(t)$.
Aproximación con 500 puntos.

$$T(\widehat{N_i}) - t_i = 0$$

En la práctica, esto nos permite asumir que disponemos de todos los valores del dominio de $N = N_{ELI}(t)$. Con este resultado podemos considerar el ajuste de los parámetros de ELI.

Dada una muestra $(t_i, N_i)$, el modelo permite obtener una predicción $\widehat{N_i} = N_{ELI}(t_i)$ de la misma; el error sólo depende de los parámetros:

$$e_i(a, k, D_0, N_0) = |N_i - \widehat{N_i}| = |N_i - N_{ELI}(t)|$$

Para el conjunto total de las muestras $D = \{(t_i, N_i)/t_i \in \mathbb{N} \wedge 0 \leq t_i \leq N_F\}$ generamos una función de error total del modelo[7]:

$$E(a, k, D_0, N_0) = \sum_{i=0}^{t_i \leq N_F} (e_i)^2$$

Dicho error total debe ser minimizado:

$$min[E(a, k, D_0, N_0)]$$

sujeto a:

$$a > 0;\ k > 0;\ D_0 > 0;\ N_0 > 0$$

[7]Observar que estamos empleando una minimización de mínimos cuadrados debido a su conveniencia analítica para la mayoría de algoritmos de minimización en dominios convexos.

El proceso de minimización nos proporcionará los parámetros óptimos del modelo $a,\ k,\ D_0$, y $N_0$. Ver la sección "Evaluación de ELI" en los anexos para la implementación de esta minimización mediante Octave.

Una vez determinados los parámetros del modelo, sus predicciones $\widehat{N_i} = N_{ELI}(t_i)$ se contrastan con los de la muestra $N_i$ a fin de obtener una idea del éxito del modelo.

### 4.2.3. Modelos Alternativos a ELI

Es importante contrastar la bondad de nuestro modelo en relación a otras posibilidades. A tal efecto, en esta sección proponemos algunas alternativas "inspiradas" en la regla de correspondencia de ELI, así como en su gráfica.

#### 4.2.3.1. Modelo Logarítmico-Lineal Directo: LLD

En tanto ELI posee la estructura (ecuación 4.2.3):

$$t = \alpha N + \beta e^{-\gamma N} + \delta$$

Entonces, una aproximación a la función implícita $N = N(t)$ consiste en considerar el factor exponencial por separado y plantear que $N(t) = b\log(ct)$; como el argumento del logaritmo no puede hacerse cero en $t = 0$ entonces se refina mediante $N(t) = b\log(1 + ct)$.

De otro lado, según 4.2.7 la tendencia en el límite $t \to \infty$ para $\frac{dN}{dt}$ es una constante, lo que sugiere introducir un término lineal con su intercepto, con lo que planteamos el modelo Logarítmico-Lineal Directo "LLD":

$$N(t) = a + b\log(1 + ct) + dt \tag{4.2.15}$$

Este modelo "compite" con ELI, y tiene la gran ventaja de estar formulado con una regla de correspondencia explícita (por lo que le hemos denominado "directo".)

#### 4.2.3.2. Modelo Exponencial-Lineal Directo: ELD

Si imaginamos la gráfica de ELI con el signo opuesto (reflejo vertical), es claro que su concavidad puede aproximarse mediante una función exponencial de la forma $N(t) = a - be^{-ct}$; por el mismo argumento que el caso anterior, se puede introducir un término lineal con lo que arribamos al modelo Exponencial-Lineal Directo "ELD":

$$N(t) = a - be^{-ct} + dt \tag{4.2.16}$$

#### 4.2.3.3. Modelos Polinomiales

A fin de disponer de mayor material de contrastación también incluimos los modelos basados en polinomios:

1. Modelo cuadrático de la forma $N(t) = at^2 + bt + c$ correspondiendo a la posibilidad más simple (aparte de la función lineal)

2. Modelo cúbico de la forma $N(t) = at^3 + bt^2 + ct + d$ que es una extensión del anterior y cuenta con el mismo número de parámetros que ELI

### 4.2.4. Ajuste de los Modelos Alternativos

Los modelos polinomiales se implementan mediante la anteriormente descrita interpolación polinomial (ecuación 2.1.3) con ayuda de la función `polyfit()` de Octave.

En el caso del modelo ELD requerimos transformarlo en un problema de optimización en forma análoga al caso ELI. Un punto inicial razonable $(a_0, b_0, c_0, d_0)$ puede ser estimado mediante algunas observaciones algebraicas. Por ejemplo, para $t = 0$ el código fuente corresponde al valor inicial $N(0) = N_0$. Para $t \to \infty$ usamos la aproximación $N(\infty) \approx N_F$. Asumiendo que la contribución lineal es pequeña, podemos asumir $d_0 = 0$; luego:

$$N(t_F = \infty) = N_F \Rightarrow N_F = a_0$$

$$N(0) = N_0 \Rightarrow N_0 = a_0 - b_0 \Rightarrow b_0 = N_F - N_0$$

Para el "punto medio" temporal, la curva ha alcanzado un valor relativamente alto (dada su convexidad); podemos asumir por tanto:

$$N(\frac{t_F}{2}) = \alpha N_F + (1 - \alpha)N_0$$

donde $\alpha$ es cercano a 1 (en nuestra aproximación empleamos "0,9".) Luego

$$\alpha N_F + (1 - \alpha)N_0 = N_F - (N_F - N_0)\exp(-\frac{c_0 t_F}{2}) \Rightarrow$$

$$c_0 = -\frac{2}{T_F}\log\left(\frac{(1 - \alpha)(N_0 + N_F)}{N_F - N_0}\right)$$

El modelo LLD también es ajustado mediante optimización, por lo que requerimos un punto inicial adecuado $(a_0, b_0, c_0, d_0)$; el parámetro para el término lineal se asume libre de signo: $d_0 = 0$. Para $t = 0$ tenemos que $a_0 = N_0$; considerando $t = T_F$:

$$N_F = N_0 + b\log(1 + c_0 T_F) \Rightarrow b_0 \log(1 + cT_F) = N_F - N_0$$

En forma análoga al caso anterior, para el "punto medio" consideramos

$$N(\frac{t_F}{2}) = \alpha N_F + (1 - \alpha)N_0 \Rightarrow$$

$$\alpha N_F + (1 - \alpha)N_0 = N_0 + b_0 \log(1 + \frac{c_0 T_F}{2}) \Rightarrow$$

$$b_0 \log(1 + \frac{c_0 T_F}{2}) = \alpha(N_F - N_0)$$

Dividiendo las últimas ecuaciones para eliminar "$b$":

$$\frac{\log(1 + c_0 T_F)}{\log(1 + \frac{c_0 T_F}{2})} = \frac{N_F - N_0}{\alpha(N_F - N_0)} = \frac{1}{\alpha}$$

aproximando $\log(1 + R) \approx \log(R)$ pues $R \gg 1$:

$$\alpha \log(c_0 T_F) = \log(\frac{c_0 T_F}{2}) \Rightarrow$$

$$(c_0 T_F)^\alpha = (\frac{c_0 T_F}{2}) \Rightarrow c_0 = \frac{2^{\frac{1}{1-\alpha}}}{T_F}$$

Hallado $c_0$ se puede emplear cualquiera de las relaciones anteriores para estimar $b_0$:

$$b_0 = \frac{N_F}{\log(1 + c_0 t_F)}$$

### 4.2.5. Contrastación Empírica de Modelos para Tendencia Ideal

#### 4.2.5.1. Métricas de Bondad de Ajuste

Una métrica muy útil para estimar la bondad del ajuste de diversas funciones (modelos) a un conjunto de datos es el NRMSE (Normalized Root Mean Squared Error.) Al ser adimensional también facilita la comparación del ajuste de un modelo específico entre distintos conjuntos de datos.

Existen diversas fórmulas para esta métrica, las cuales varían de acuerdo al término de normalización (el denominador); entre éstas hemos optado por[8]:

[8]Otras fórmulas alternativas (mencionadas en nuestro anexo "Medidas de Capacidad Predictiva") proporcionan resultados muy similares para nuestros datos (salvo una constante de proporcionalidad.) Acerca del divisor de normalización, consideramos que uso de la diferencia máx(.) − mín(.) puede sobre-representar valores extremos de la muestra; asimismo, el uso del

$$\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}}{StdDev(X)}$$

Donde $StdDev(.)$ corresponde a la desviación estándar del conjunto de datos observados, es decir, los valores $N_i$ correspondientes al número de líneas de código fuente para cierta semana.

Del mismo modo, MAE (Mean Absolute Error) es otra métrica comunmente empleada en la contrastación de modelos de estimación de esfuerzo, cuya definición es:

$$MAE = \frac{\sum_{i=1}^{n} |\widehat{x_i} - x_i|}{n}$$

Para nuestros fines, MAE proporciona un estimado del error "promedio" en la predicción de las líneas de código del proyecto. Observar que su dimensión precísamente son las líneas de código fuente, por lo que su magnitud tiene una interpretación directamente asociada a los costos del proyecto, o más precisamente, a los sobrecostos.

#### 4.2.5.2. Extracción de Datos para Tendencia Ideal

A partir de la noción propuesta para la tendencia ideal, hemos considerado la acumulación semanal de las líneas de código fuente tal como señalamos anteriormente. Sin embargo, la elección del intervalo temporal durante el cual asumiremos que la tendencia ideal es manifiesta (o en todo caso, es la que se manifiesta con preeminencia) es relativamente arbitraria. Al respecto hemos seguido los siguientes criterios heurísticos:

**Paso 1** A partir del examen de la gráfica de productividad, determinamos la semana de máximo rendimiento, la cual normalmente estará al principio del proyecto por los motivos explicados anteriormente. Consideramos que todo lo anterior corresponde a actividades de "prototipo".

**Paso 2** La gráfica de productividad decae hasta un punto "cercano a cero" (normalmente menor al 5 % del valor del "pico inicial".) Este punto es considerado el final de la influencia de la tendencia ideal.

**Paso 3** Tras el fin de la tendencia ideal hallado en el paso 2, requerimos que discurran al menos cuatro semanas sin actividad altamente significativa (de más

valor promedio podría generar distorsiones para muestras que están muy alejadas del cero (en términos relativos) versus aquellas que son muy cercanas.

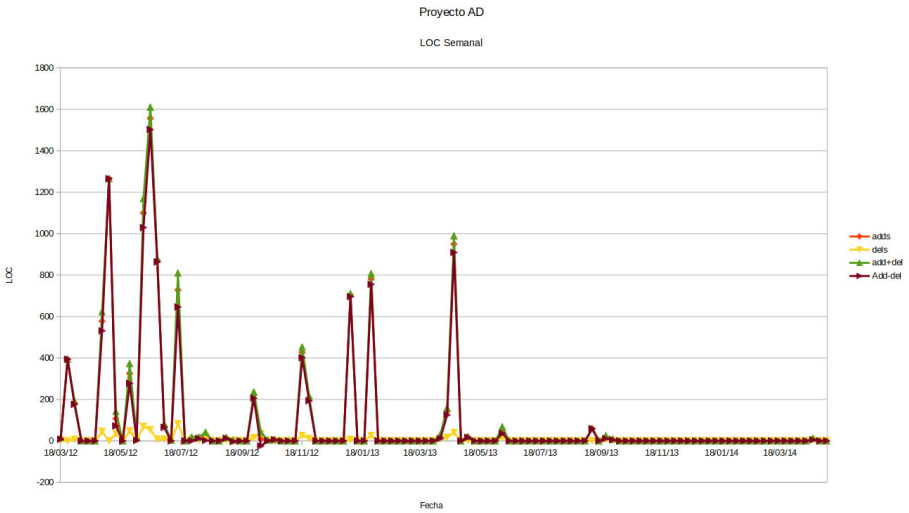



Figura 4.2.3: LOC semanal proyecto “AD”.

| Proy. | ELI | CUAD | CUB | ELD | LLD | #Sem. |
|---|---|---|---|---|---|---|
| SP | 8.70 | 11.47 | 8.42 | 48.15 | 6.75 | 58 |
| RE | 33.36 | 28.36 | 10.12 | 8.13 | 15.06 | 14 |
| AD | 19.39 | 17.36 | 15.53 | 26.00 | 18.96 | 16 |
| AS | 23.56 | 22.94 | 22.77 | 22.99 | 23.35 | 16 |
| JP | 22.62 | 14.76 | 9.99 | 15.60 | 18.33 | 7 |
| CT | 32.08 | 17.28 | 5.20 | 6.51 | 7.61 | 15 |
| PO | 42.51 | 20.41 | 8.65 | 27.17 | 29.04 | 9 |

Cuadro 4.2: NRMSE para Tendencia Ideal

de 30 % del “pico anterior”.) Si esto no ocurre, asumimos que la tendencia ideal no ha concluido y se retorna al paso 2 partiendo desde la última semana con actividad. Esta iteración normalmente sólo se da en proyectos con un patrón temporal errático de actividades.

Notablemente, el proyecto “AD” exige algunas iteraciones de el paso 3 dado su patrón con interrupciones (figura 4.2.3.)

#### 4.2.5.3. Resultados NRMSE

La tabla 4.2 proporciona los valores de NRMSE para los siete proyectos considerados en relación al código “neto” obtenido como la diferencia entre las líneas añadidas y las líneas eliminadas del código fuente. La última columna proporciona el número de muestras (semanas) incluidas en el ajuste.

Notar que las aproximaciones “polinómicas” claramente superan al resto de modelos, lo que ocurre por su capacidad de amoldarse con menos restricciones

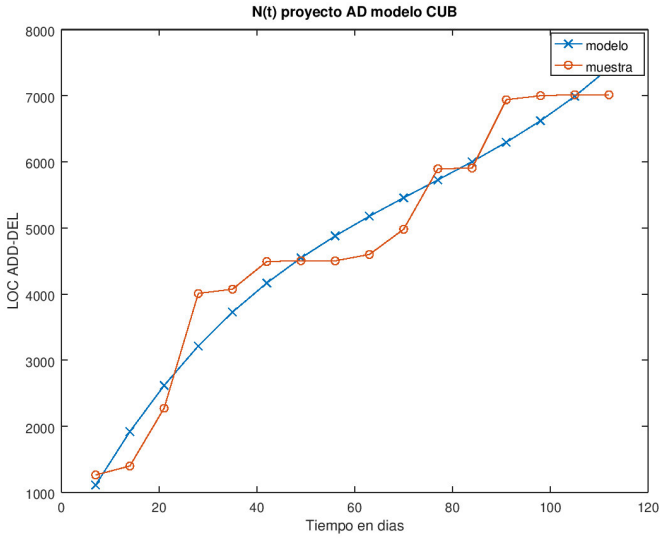



Figura 4.2.4: Ajuste proyecto “AD” con modelo cúbico.

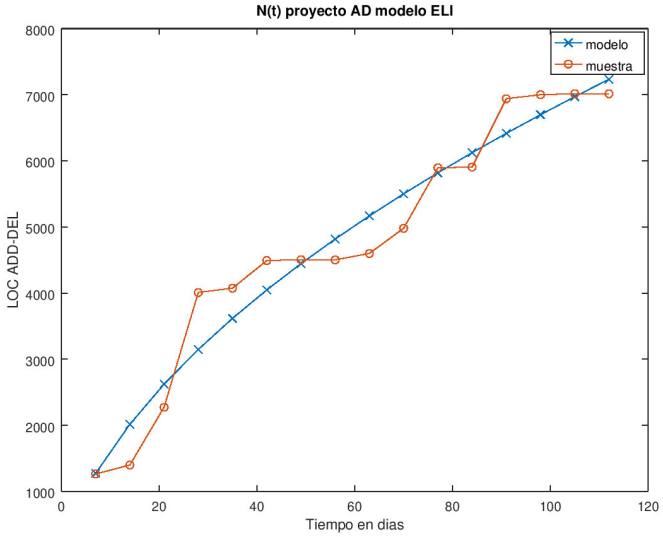



Figura 4.2.5: Ajuste proyecto “AD” con modelo ELI.

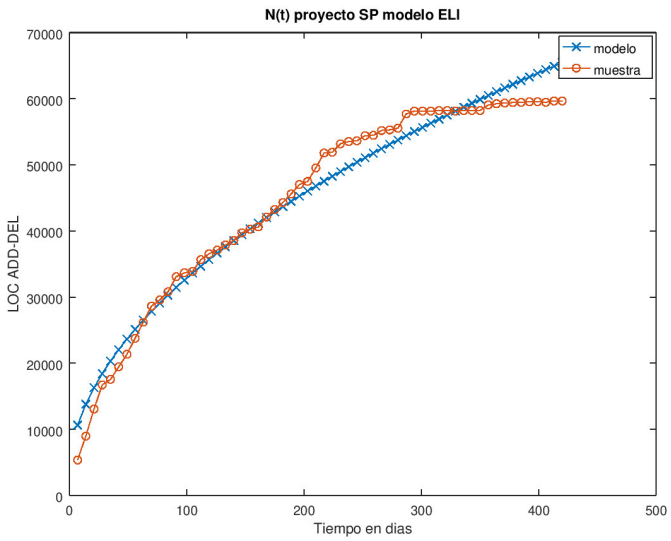



Figura 4.2.6: Ajuste proyecto “SP” con modelo ELI.

morfológicas en relación a las funciones de clase exponencial o logarítmica del resto de modelos.

Notablemente el proyecto SP presenta ajustes sorprendentemente buenos (de acuerdo a la inspección gráfica) lo que es especialmente significativo dado su elevado número de muestras. En las figuras 4.2.4, 4.2.5, 4.2.6 y 4.2.7 hemos graficado los ajustes de algunos modelos sobre las muestras de dos proyectos considerados al azar; notar que los cinco modelos aplicados sobre nuestros siete proyectos totalizan 35 ajustes, los que se han graficado por cuestiones de espacio.

#### 4.2.5.4. Resultados MAE

La tabla 4.3 proporciona los valores de MAE para los proyectos y modelos considerados. Estos valores deben emplearse para contrastar la capacidad de los modelos para cada proyecto (por el contrario, no deben emplearse para la comparación de ajustes inter-proyecto debido a su falta de normalización.) Tienen un correlato directo con la realidad (específicamente, con los costos) puesto que corresponden a líneas de código fuente que -en promedio- el modelo falla en estimar (por exceso u omisión.)

En términos generales, es de esperarse que en un proyecto más grande (en este caso, durante una fase inicial más prolongada) el fallo en la estimación tienda a crecer en la medida que el modelo tiene más oportunidades de perder validez. Visto en sentido contrario, si un proyecto presenta un MAE elevado a pesar de ser más breve que otro que sirve de referencia, significa que tiene problemas en su ejecución y el uso de sus recursos.

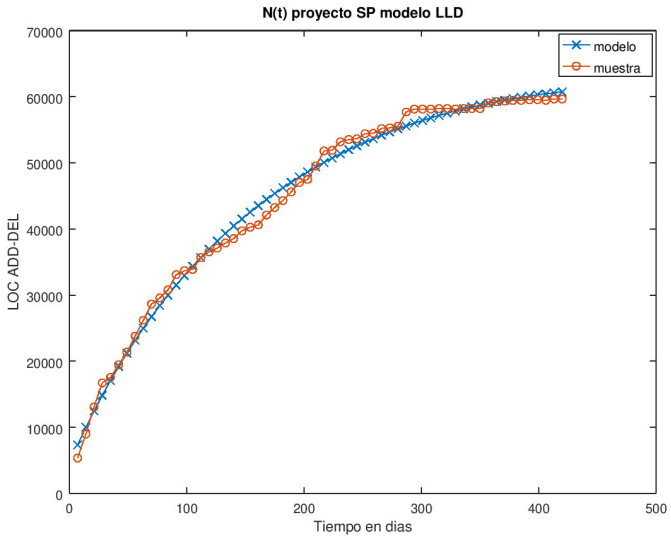



Figura 4.2.7: Ajuste proyecto "SP" con modelo LLD.

| Proy. | ELI | CUAD | CUB | ELD | LLD | #Sem. |
|---|---|---|---|---|---|---|
| SP | 1003.20 | 964.90 | 989.44 | 2790.10 | 1097.23 | 58 |
| RE | 1155.00 | 1060.90 | 330.82 | 288.30 | 579.68 | 14 |
| AD | 395.72 | 340.59 | 329.44 | 532.18 | 386.67 | 16 |
| AS | 160.71 | 159.27 | 152.04 | 160.03 | 159.34 | 16 |
| JP | 75.64 | 52.73 | 31.14 | 56.67 | 69.19 | 7 |
| CT | 526.27 | 280.23 | 81.49 | 98.34 | 118.19 | 15 |
| PO | 1406.16 | 663.06 | 269.11 | 869.86 | 804.31 | 9 |

Cuadro 4.3: MAE para Tendencia Ideal

| Proy. | ELI | CUAD | CUB | ELD | LLD |
|---|---|---|---|---|---|
| SP | 3140.75 | 1855.82 | 2049.55 | 3990.21 | 2715.13 |
| RE | 1908.76 | 2190.17 | 3995.80 | 5390.86 | 3955.27 |
| AD | 686.62 | 798.04 | 1074.75 | 1524.69 | 691.72 |
| AS | 197.24 | 221.00 | 189.05 | 224.45 | 204.47 |
| JP | 280.17 | 334.71 | 238.38 | 340.42 | 349.92 |
| CT | 780.35 | 986.16 | 1454.10 | 1600.44 | 1654.47 |
| PO | 1861.09 | 3335.87 | 4738.92 | 3670.26 | 2527.16 |

Cuadro 4.4: Productividad en $t_0$

| Proy. | ELI | CUAD | CUB | ELD | LLD |
|---|---|---|---|---|---|
| SP | 557.19 | -143.24 | 50.50 | 44.76 | 222.19 |
| RE | 638.79 | -379.28 | 1426.35 | 409.54 | 103.75 |
| AD | 375.70 | 202.06 | 478.77 | 81.44 | 333.97 |
| AS | 136.82 | 106.76 | 74.81 | 112.85 | 130.03 |
| JP | 133.34 | 24.71 | -71.62 | 37.58 | 75.01 |
| CT | 258.21 | -245.97 | 221.98 | 39.15 | -8.84 |
| PO | 834.32 | -1033.07 | 369.98 | 549.24 | -126.57 |

Cuadro 4.5: Productividad en $t_F$

#### 4.2.5.5. Resultados para la Productividad

Una vez obtenidos los modelos, resulta sencillo obtener la productividad $N'(t)$, la cual es aproximada mediante:

$$N'(t) \approx \frac{N(t+1) - N(t)}{\Delta t} = N(t+1) - N(t)$$

El denominador es $\Delta t = 1$ semana, lo que significa que la productividad está expresada en líneas de código por semana. La tabla 4.6 proporciona los valores de $N'(t_0)$ ajustados para cada modelo; la tabla 4.5 hace lo propio para $N'(t_F)$.

Los modelos con productividad final negativa son altamente dudosos, especialmente porque tal negatividad no es consistentemente manifiesta para ninguno de los proyectos; claramente se trata de un efecto indeseable producto del ajuste de la curva al intentar minimizar el error desde el punto de vista exclusivamente numérico; desde este punto de vista el modelo ELI resulta ser el más confiable (seguido por ELD.)

Sólo considerando a ELI, resulta interesante analizar la caída de productividad de los proyectos mediante una simple variación porcentual:

$$\Delta P = \frac{N'(t_0) - N'(t_F)}{N'(t_0)} 100\,\%$$

Los resultados para la caída de productividad se proporcionan en la tabla 4.6.

Dicha caída de productividad es un rasgo inherente al proyecto en el que

| Proyecto | $\Delta P$ |
|---|---|
| SP | 82.26 |
| RE | 66.53 |
| AD | 45.28 |
| AS | 30.63 |
| JP | 52.41 |
| CT | 66.91 |
| PO | 55.17 |

Cuadro 4.6: Caída de Productividad en Tendencia Ideal.

| ELI | CUAD | CUB | ELD | LLD |
|---|---|---|---|---|
| 26.97 | 18.59 | 11.56 | 18.31 | 17.27 |

Cuadro 4.7: Promedio NRMSE

confluye la complejidad del problema así como el esfuerzo total aplicado. Como regla general, observamos que los proyectos más grandes (en tamaño) presentan una mayor caída de productividad a lo largo de su desarrollo.

### 4.2.6. Contraste de Modelos de Tendencia Ideal

#### 4.2.6.1. Evaluación estadística de calidad de ajuste

Debemos señalar que no existe un claro modelo ganador bajo los criterios considerados; por ejemplo, promediando los valores NRMSE de cada modelo, el modelo "cúbico" parece ser el mejor desde el punto de vista de esta métrica (tabla 4.7), pero debemos tener en cuenta que el modelo cúbico (así como el cuadrático) es esencialmente inválido puesto que:

- Carece de un planteamiento teórico que lo sustente
- No presenta tendencia asintótica lineal con $t \to \infty$
- No está restringido a ningún patrón de monotonía y/o convexidad

En nuestra opinión es la última "cualidad" la que permite que el modelo cúbico minimice mejor el error en nuestras muestras: se "amolda" a las distorsiones de la tendencia ideal que presentan los datos de algunos de los casos.

Desde este punto de vista, consideramos que para efectos prácticos, el modelo LLD resulta ser un buen compromiso entre "sustento teórico" y flexibilidad, al tener una regla de correspondencia explícita y un poder de predicción comparable (ligeramente superior en nuestras muestras) a ELI.

La falta de evidencia para señalar un modelo ganador absoluto se puede justificar a partir de un enfoque estadístico. Para esto podemos intentar comparar

los valores MAE generados por los modelos, teniendo en consideración que vienen dados en unidades de líneas de código fuente; esto es, son susceptibles de ser llevados a una escala nominal. Como se trata de modelos con muestras emparejadas (por proyecto), la prueba más usual es la "t de Student"; sin embargo, dicha prueba exige supuestos que aquí no son satisfechos:

a) no tenemos justificación para asumir una distribución normal para los valores MAE (de hecho, es más plausible que éstos valores tengan un fuerte sesgo)

b) el número de muestras es extremadamente pequeño (contamos con siete proyectos.)

Las pruebas basadas en métodos no paramétricos proporcionan una alternativa, si bien de caracter parcial; bajo las condiciones de desconocimiento de la distribución subyacente, y de reducido número de muestras, emplearemos la prueba de Wilcoxon de rango con signo (Corder y Foreman, 2014, p.40).

Metodológicamente, este análisis se justifica a partir de nuestra hipótesis general, a saber, la dinámica de proyectos evoluciona cuantitativamente a partir del código fuente, en tanto hemos planteado sendos modelos para la medición del código fuente que pretenden modelar la dinámica temporal; a fin de estudiar la bondad relativa de los mismos, planteamos la hipótesis auxiliar nula:

**H0** "Las distribuciones (normales) de los valores MAE generadas por los modelos A y B tienen las mismas medias ($\mu_D = 0$)."

La hipótesis alternativa corresponde a que la diferencia de medias no es cero ($\mu_D \neq 0$.)

Como nuestras parejas de muestras son siete (y no hay casos con diferencia igual a cero), entonces al consultar la tabla de valores críticos para la estadística T (Corder y Foreman, 2014, p.244) se encuentra el valor crítico $T_{crit} = 2$ para el caso $\alpha \leq 0{,}05$ que corresponde a la significancia usualmente buscada.

En tanto Octave no proporciona un test Wilcoxon para muestras pequeñas (el mínimo exigido por su procedimiento es de 25), hemos implementado una rutina -denominada `little_wilcoxon()`- exclusivamente para el caso $n = 7$ y $T_{crit} = 2$. A continuación se muestra una captura de sesión donde se emplea esta rutina para nuestros resultados.

```
octave:59> ELI=MAE(:,1);
octave:60> ELD=MAE(:,4);
octave:61> CUB=MAE(:,3);
octave:62> LLD=MAE(:,5);
octave:64> little_wilcoxon(ELI,LLD)
T=1
Se rechaza hipotesis nula
octave:65> little_wilcoxon(ELI,CUB)
```

```
T=3
No se rechaza hipotesis nula
octave:67> little_wilcoxon(CUB,ELD)
T=8
No se rechaza hipotesis nula
octave:68> little_wilcoxon(ELI,ELD)
T=13
No se rechaza hipotesis nula
octave:69> little_wilcoxon(LLD,CUB)
T=3
No se rechaza hipotesis nula
octave:70> little_wilcoxon(LLD,ELD)
T=14
No se rechaza hipotesis nula
```

En prácticamente todos los casos no es posible rechazar la hipótesis nula, salvo en la comparación ELI vs LLD. Sin embargo, LLD no resulta ser suficientemente superior a los otros modelos.

#### 4.2.6.2. Predicción de la Duración del Proyecto

En esta sección intentamos emplear los resultados anteriores para extrapolar el comportamiento de un proyecto, y específicamente para predecir su fecha de finalización a partir la experiencia acumulada en proyectos similares. Esto se hará a partir de la información registrada para las primeras semanas de su ejecución.

Notar que esto no corresponde a un método de estimación de esfuerzo como los reseñados en los antecedentes de este estudio, los cuales operan a partir de información recolectada previamente al inicio de la codificación. En lugar de una estimación, aquí intentaremos una extrapolación de las tendencias.

El punto de partida natural para la predicción de la duración es el tamaño total del código fuente. Al respecto, disponemos de las técnicas reseñadas en los antecedentes así como una alternativa presentada en la sección 4.2.6.3. Asumiéndose que por alguno de estos métodos se dispone de un estimado para el tamaño final del código fuente $N(t_F)_{est}$, estimaremos la duración $t_F$ del proyecto asociado.

Como señalamos, esto se puede llevar a cabo mediante la extrapolación de la información recogida en las primeras semanas del proyecto, para lo cual es suficiente obtener la estimación ELI del código fuente semana a semana hasta alcanzar el tamaño que se asumió como final:

$$N_i \geq N(T_F)_{est} \Rightarrow T_F \approx i$$

El algoritmo 4.1 detalla esta idea. La tabla 4.8 ilustra los resultados de este método de predicción empleando los valores exactos para el tamaño de las muestras de la tabla 3.4. Notar que el algoritmo puede fallar al no encontrarse una

**Algoritmo 4.1** Predicción de tendencia ideal

```
Entradas:
TLOC: Tamaño final del código fuente
LOC: Vector con líneas de código fuente para las primeras semanas
SLIM: Limite admisible de semanas

Salida: Duración tentativa de proyecto, o error.

Funciones auxliares:
FIT_ELI(LOC): Ajuste de parámetros de modelo ELI
ELI(PARAM,SEMANA): Predicción LOC del modelo ELI
LEN(LOC): Longitud de vector

PASO 1: Ajustar parámetros modelo ELI a partir data LOC
  PARAMS <- FIT_ELI(LOC)
  SEMANA <- LEN(LOC)

PASO 2: Obtener tamaño según ELI
  CURRENT <- ELI(PARAMS,SEMANA)

PASO 3: Validar si se alcanza el tamaño
  Si CURRENT >= TLOC : mostrar semana y terminar
  Si SEMANA >= SLIM: terminar con error
  SEMANA <- SEMANA + 1
  Volver a PASO 2
```

| Proyecto | 4 semanas | 6 semanas | 8 semanas | Real |
|---|---|---|---|---|
| SP | 16 | 22 | 39 | 58 |
| RE | 6 | 8 | 10 | 14 |
| AD | 14 | 12 | N/A | 16 |
| AS | 40 | 14 | 13 | 16 |
| JP | 5 | 6 | N/A | 7 |
| CT | 7 | 8 | 9 | 15 |
| PO | 5 | 6 | 7 | 9 |

Cuadro 4.8: Predicciones (en semanas) para tendencia ideal usando tamaño final real. Se incluyen predicciones con distintos tamaños de muestra inicial.

predicción para un lapso razonable, lo que apunta a una deficiencia del modelo o a un fuerte cambio en el patrón de programación del proyecto a medio camino. En las figuras 4.2.8 hasta 4.2.19 se puede visualizar gráficamente la predicción del modelo ELI a partir de la información de las primeras semanas.

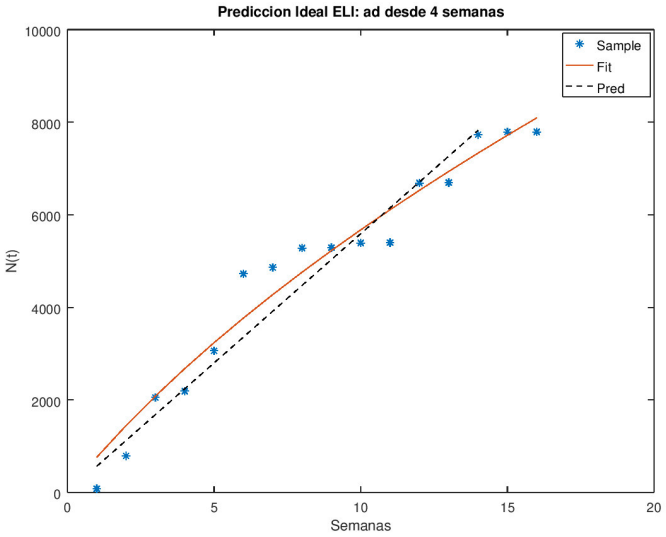



Figura 4.2.8: Predicción AD via tendencia ideal.

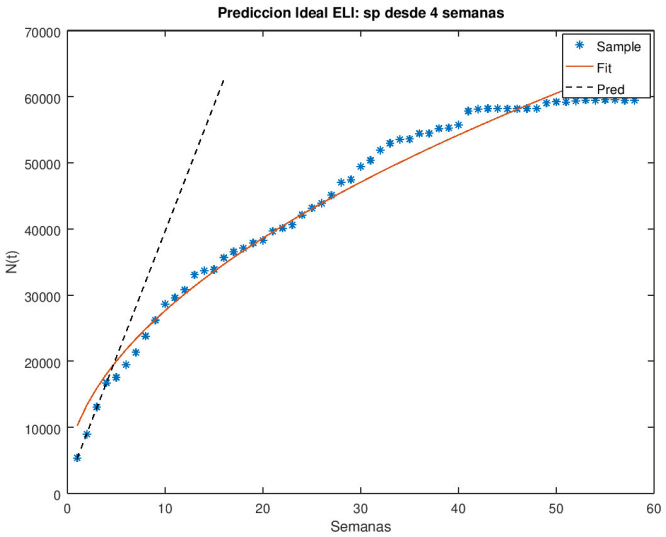



Figura 4.2.9: Predicción SP via tendencia ideal desde 4 semanas.

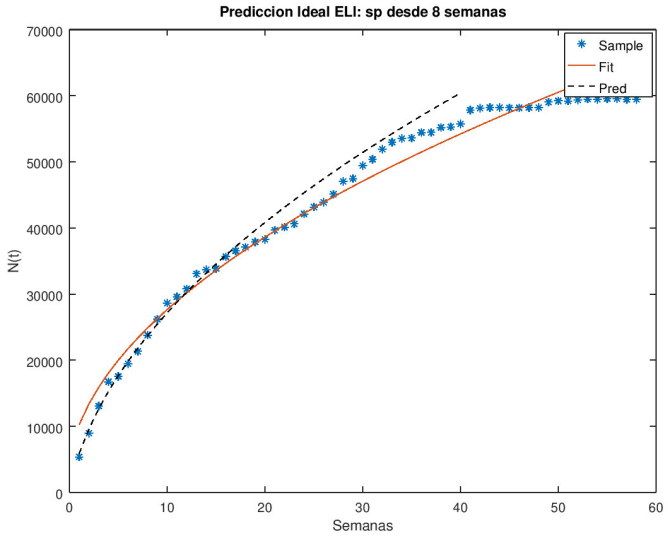



Figura 4.2.10: Predicción SP via tendencia ideal desde 8 semanas.

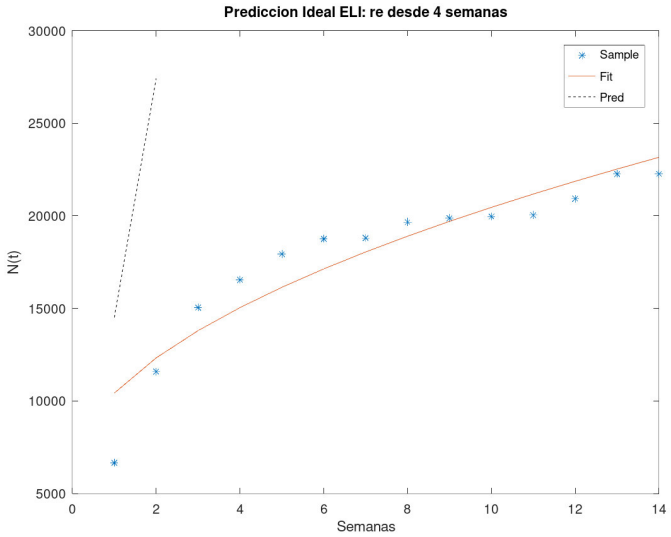



Figura 4.2.11: Predicción RE via tendencia ideal desde 4 semanas.

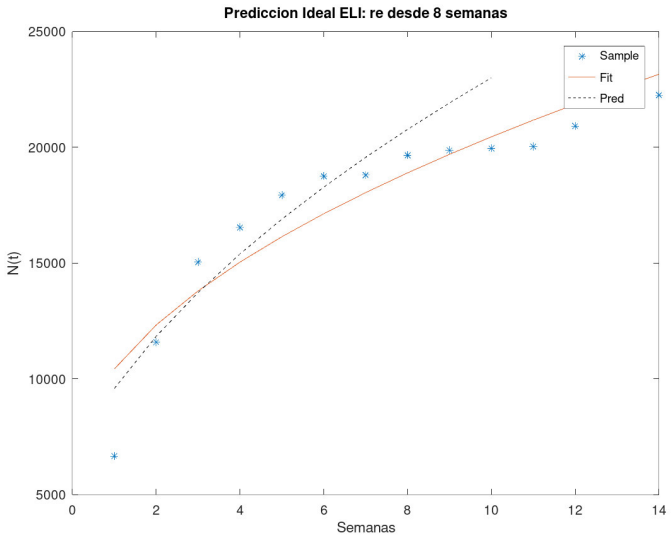



Figura 4.2.12: Predicción RE via tendencia ideal desde 8 semanas.

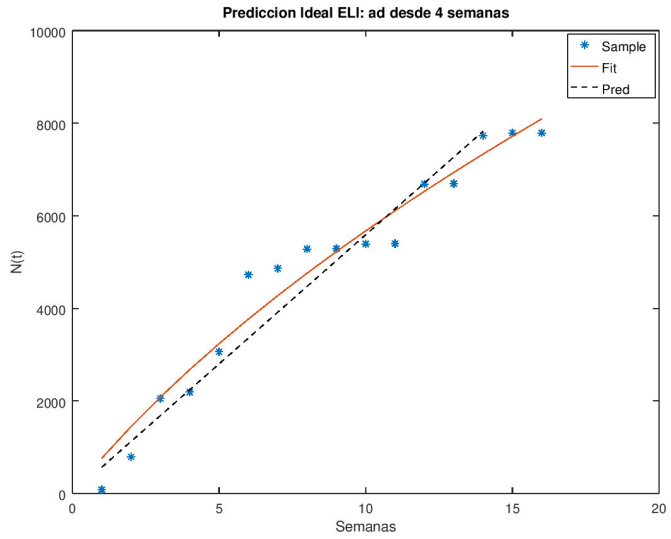



Figura 4.2.13: Predicción AS via tendencia ideal desde 4 semanas.

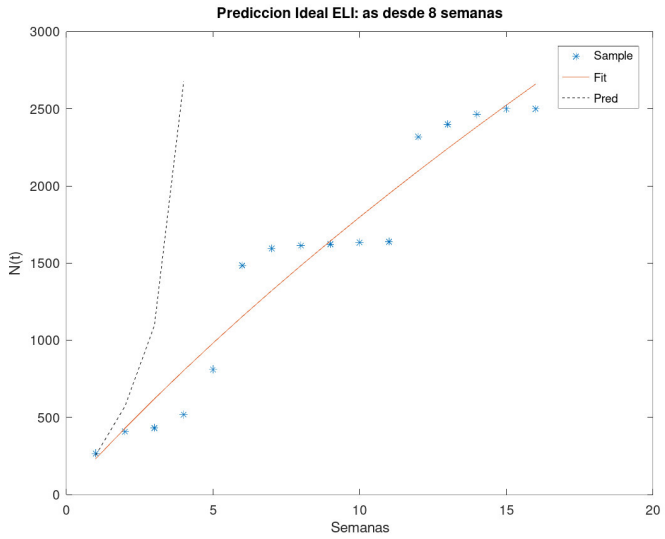



Figura 4.2.14: Predicción AS via tendencia ideal desde 8 semanas.

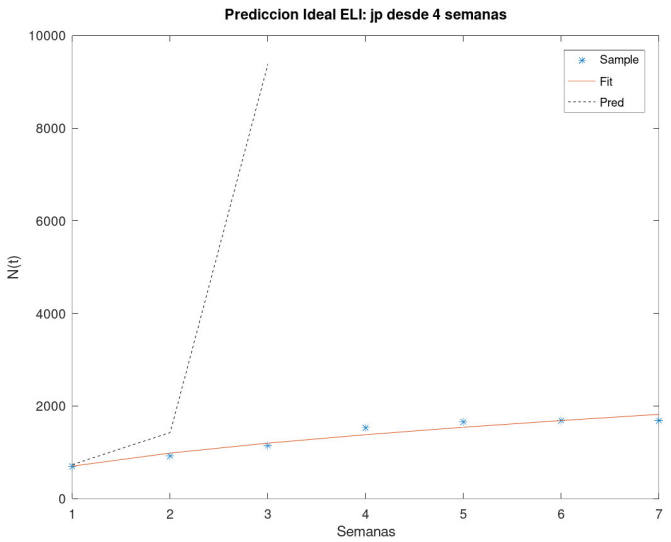



Figura 4.2.15: Predicción JP via tendencia ideal desde 4 semanas.

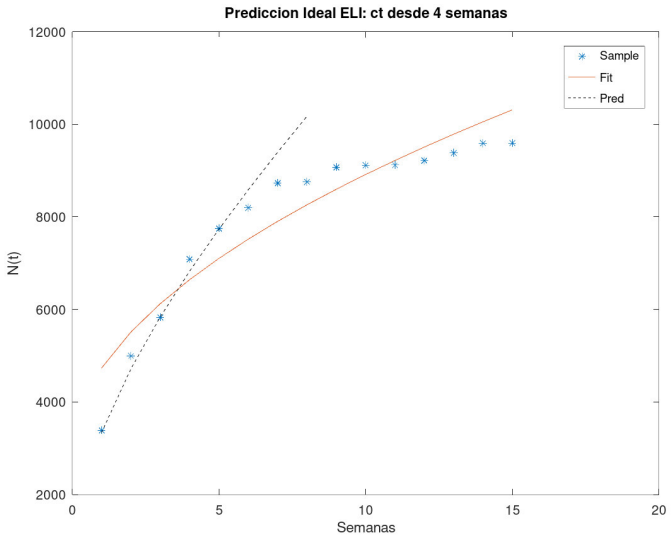



Figura 4.2.16: Predicción CT via tendencia ideal desde 4 semanas.

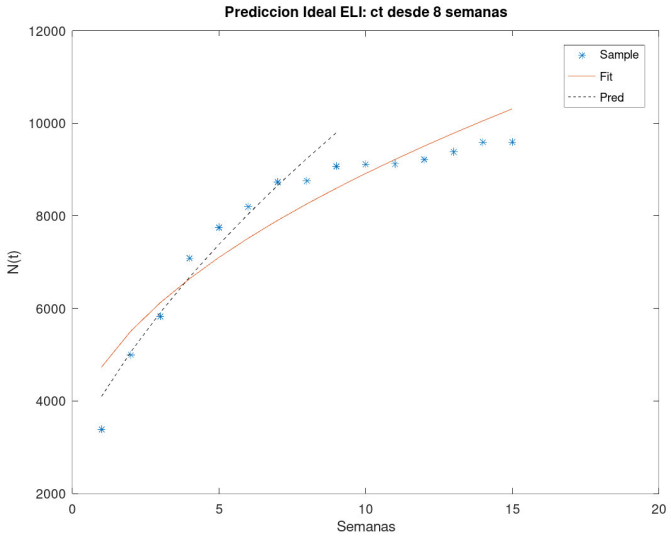



Figura 4.2.17: Predicción CT via tendencia ideal desde 8 semanas.

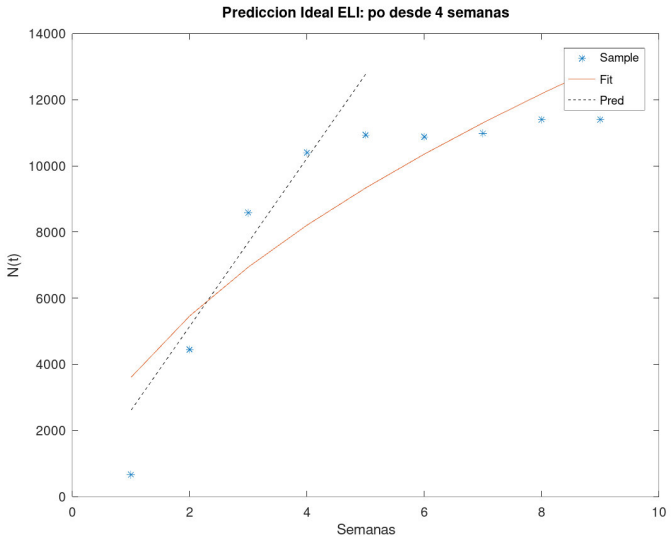



Figura 4.2.18: Predicción PO via tendencia ideal desde 4 semanas.

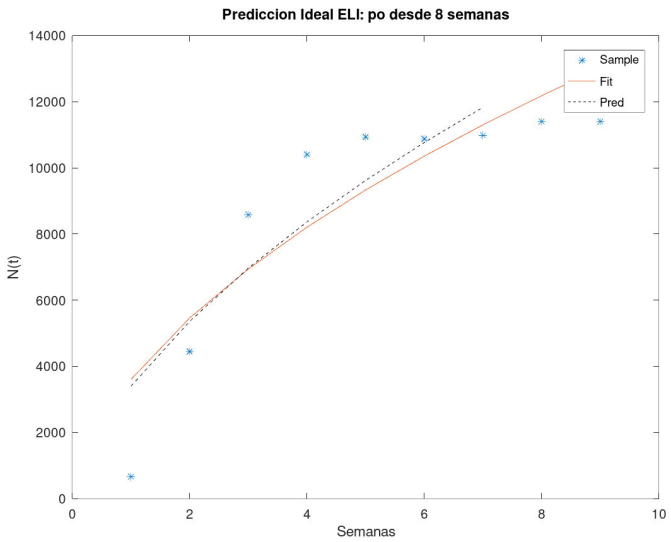



Figura 4.2.19: Predicción PO via tendencia ideal desde 8 semanas.

| Proyecto | 4 semanas | 6 semanas | 8 semanas | Real |
|---|---|---|---|---|
| SP | 34 | 92 | 228 | 58 |
| RE | 72 | N/A | N/A | 14 |
| AD | 9 | 9 | N/A | 16 |
| AS | 42 | 12 | 12 | 16 |
| JP | 19 | 49 | N/A | 7 |
| CT | 83 | 311 | N/A | 15 |
| PO | 15 | 36 | 119 | 9 |

Cuadro 4.9: Predicciones (en semanas) para tendencia ideal usando tamaño final estimado. Se incluyen predicciones con distintos tamaños de muestra inicial.

#### 4.2.6.3. Estimación del tamaño final del código fuente

Esta sección se inicia con una observación empírica: los proyectos cuya productividad inicial es elevada tienden a tener un mayor tamaño. Este hecho puede justificarse a partir de los argumentos de "dificultad" discutidos en torno a la ecuación 4.2.1: los proyectos que tienen mucho código fuente por implementar, tendrán (en promedio) una mayor cantidad de código de muy baja dificultad que los que tienen poco código a implementar; luego, la productividad inicial (que es el momento donde aquél es atendido) será mayor. Esto sugiere un modelo de aproximación empírico:

$$N(t_F) \approx G[N'(t_0)]$$

A partir de los datos disponibles para $N(t_F)$ y $N'(t_0)$ (tablas 4.4 y 3.4) se plantea un modelo lineal[9] que permite estimar el tamaño final del código fuente conociéndose la productividad inicial:

$$\sqrt{\widehat{N(t_F)}} = \beta_1 + \beta_2 N'(t_0)$$

Para nuestros datos, el ajuste mediante regresión lineal genera la siguiente relación:

$$\widehat{N(t_F)} = [33{,}89 + 0{,}06097 N'(t_0)]^2$$

Donde se obtuvo además que $R^2 = 0{,}911$. El uso de esta aproximación reduce la calidad de la predicción en relación al uso del tamaño final exacto; sin embargo, consideramos que puede servir como punto de partida o una referencia adicional para los responsables del proyecto. Comparar las tablas 4.8 y 4.9.

[9]Los modelos lineales $N(t_F) = \beta_1 + \beta_2 N'(t_0)$ y $\log[N(t_F)] = \beta_1 + \beta_2 N'(t_0)$ generaron predicciones marcadamente deficientes. Sus coeficientes de determinación son respectivamente $R^2 = 0{,}834$ y $R^2 = 0{,}8667$.

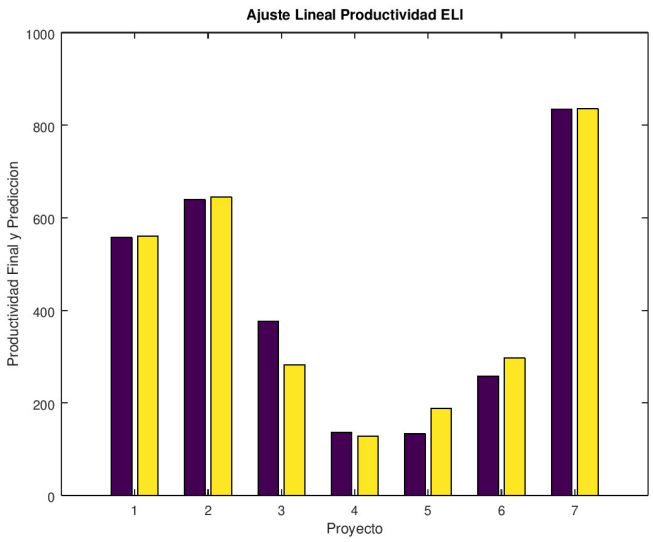



Figura 4.2.20: Productividad ELI final de proyectos y predicción. Productividad ELI en color morado; predicción según modelo de la ecuación 4.2.18 en amarillo.

#### 4.2.6.4. Relaciones de Productividad

Esta sección releva una observación auxiliar descubierta a partir de los datos procesados, no siendo un componente esencial de la propuesta de este trabajo.

Tal como vimos en la sección 4.2.5.5, la productividad final posee cierta tendencia relacionada con la productividad inicial así como el tamaño total del software. Al respecto propondremos un modelo lineal para relacionar la productividad final de un proyecto a partir de la productividad inicial y el tamaño final del código fuente:

$$N'(t_F) = \beta_1 + \beta_2 N'(T_0) + \beta_3 LOC \tag{4.2.17}$$

El ajuste mediante regresión lineal (sección 2.1) proporciona los coeficientes del modelo con un elevado coeficiente de determinación de $R^2 = 0{,}969$:

$$N'(t_F) = 72{,}57 + 0{,}531 N'(T_0) - 0{,}01988 LOC \tag{4.2.18}$$

Esta ecuación puede utilizarse para explorar las relaciones de productividad y el tamaño final del código fuente. Por ejemplo, un valor extremadamente reducido de productividad final implica una gran lentitud en el avance de la codificación en sus últimas semanas (con la ineficiencia y baja de moral que esto suele implicar), lo que sugiere la necesidad de apuntalar el equipo (para elevar la "productividad inicial" de una nueva fase), o un replanteamiento del alcance del proyecto que reduzca su tamaño.

## 4.3. Modelamiento de la Tendencia Estocástica Procesual

### 4.3.1. Planteamiento Analítico de Tendencia Estocástica

Tal como vimos en la sección 1.2.3.2, resulta razonable considerar los cambios en el código fuente como eventos de un proceso estocástico en la medida que aquellos ocurren con un patrón fuertemente aleatorio; los autores del mencionado estudio equiparan estos eventos con un proceso de Poisson no homogéneo (NHPP) pues la tasa de ocurrencia de los mismos varía considerablemente a lo largo de la vida de cualquier proyecto.

Coincidentemente, este patrón es muy empleado para modelar los fallos de ejecución de los productos de software (de proyectos terminados), lo que hace plausible equiparar a los requerimientos de un proyecto (que aún no están implementados) con defectos en el producto (el defecto corresponde a la ausencia de la funcionalidad requerida.)

A fin de hacer uso del modelamiento de la fiabilidad de software, el proceso de Poisson exige el conteo de fallos como unidad de medida. En el ya mencionado estudio de Aman y Ohkochi (2010), se consideran proyectos de "código abierto" de larga duración (de muchos meses) donde una comunidad de programadores realiza cambios cada cierto tiempo:

Consideramos un cambio de código fuente como un evento estocástico puesto que muchos programadores continuarían registrando el código en el repositorio concurrentemente; en otras palabras, sería incierto cuándo ocurriría el siguiente cambio en el código. (Aman y Ohkochi, 2010, p.47)

Para fines del ajuste NHPP dicho estudio sólo considera la existencia de los cambios en el código fuente (y no su magnitud), los cuales deben explicar totalmente la curva (acumulativa); sin embargo, en nuestro modelo hemos considerado la separación de una tendencia ideal de otra estocástica; esto significa que los "eventos" a ser considerados sólo tienen como función explicar *la discrepancia* entre la tendencia ideal y la realidad (muestra disponible), mientras que para Aman y Ohkochi (2010) la tendencia estocástica debe explicar totalmente la realidad.

Con estas consideraciones, planteamos nuestra hipótesis de trabajo:

"La tendencia estocástica se manifiesta en picos de productividad ocasionados por replanteamientos en la codificación del proyecto."

Estos picos se manifiestan en una tasa de crecimiento inusual de la tendencia de productividad, en relación a la tendencia de productividad "promedio"; en tanto la productividad a escala diaria es fuertemente cambiante, es razonable modelar su tendencia mediante el "promedio móvil simple" de la misma (moving average,

ver sección 2.5), y definir los eventos temporales de "pico" como aquellos en los que el crecimiento de dicha tendencia sobrepasa un umbral para un determinado día:

$$PICOS = \{t_i / \Delta MA_i > r_0\} \tag{4.3.1}$$

donde $\Delta MA_i$ es la variación del promedio móvil para el día $i$:

$$\Delta MA_i = MA(t_i) - MA(t_{i-1})$$

y $MA(t)$ es la función de promedio móvil para la productividad $N'(t)$ considerando el promedio de tres días (ver sección 3.4.2):

$$MA(t) = \frac{N'(t) + N'(t-1) + N'(t-2)}{3}$$

El umbral $r_0$ es un parámetro especificado empíricamente que señala a partir de qué tasa de crecimiento de productividad consideramos la presencia de un pico. Para el parámetro $r_0$ hemos empleado sistemáticamente el valor $r_0 = 10$ puesto que empíricamente observamos que valores menores producen resultados similares, mientras que valores muy elevados sólo incluirían "eventos de replanteamiento" correspondientes a valores muy extremos en la tasa de crecimiento.

Todo lo anterior nos proporciona la información temporal necesaria para el ajuste del modelo estocástico; sin embargo, es menester proporcionar la variable de conteo asociada a los eventos (por ejemplo, para el análisis de fiabilidad esto corresponde al número de fallos detectados para el lapso considerado.) A tal efecto, es razonable asumir que el conteo es proporcional a la magnitud del "pico" (es decir, un pico grande equivale a un número elevado de fallos.)

Experimentalmente hemos hallado que la relación:

$$CNT_i = \lfloor \log(\Delta MA_i) \rfloor$$

permite obtener ajustes razonables para algunos modelos NHPP, donde "razonable" se refiere a indicadores de calidad de ajuste cercanos a los que publicados para estudios de fiabilidad de software[10].

Desde el punto de vista de la ecuación 4.1.1, se ha identificado la función $h$ con el logaritmo.

Una vez definida la extracción de los datos que alimentan el modelo, debemos

[10]Otra justificación para esta propuesta corresponde a la capacidad "predictiva" de las funciones "ajustadas" con muestras parciales con respecto al proyecto total. Una vía de investigación interesante corresponde al empleo de otras familias de funciones o a su búsqueda vía criterios de optimización. Nuestro primer intento (fallido) correspondió a la más elemental $CNT_i = 1$, que equivale a identificar $h$ (de 4.1.1) con la función constante 1, descartando efectivamente la variabilidad de la magnitud de los picos.

completar la formulación del mismo. Para esto debemos considerar la función de la "media" del proceso NHPP $m(t)$ para la que existe una larga experiencia en el contexto de la fiabilidad de software, lo que nos servirá como referencia al momento de analizar la bondad de nuestra propuesta; al respecto hemos considerado pertinente el uso del modelo de Goel-Okumoto (descrito en la sección 1.2.3.1) por ser uno de los más sencillos al tiempo que ofrece resultados competitivos con sus pares más sofisticados (Sharma y col., 2010).

El uso de nuestros modelos propuestos ELI, ELD y LLD para modelar la tendencia estocástica se justifica en tanto que cuanto más código fuente es elaborado, surgen más ocasiones en que este debe ser (de algún modo) replanteado, dando lugar a los mencionados picos de desarrollo.

Como veremos, el resultado de la aplicación de estos modelos no es concluyente; sin embargo, consideramos digno de resaltar que estos modelos resultan ser "competitivos" tanto en la tendencia ideal como en la estocástica.

### 4.3.2. Ajuste de Parámetros para Tendencia Estocástica

En la estimación de parámetros para los modelos basados en distribuciones de probabilidad de la familia exponencial, se suele emplear el criterio de máxima verosimilitud lo que equivale a un problema de optimización. Sin embargo, para algunos casos particulares de procesos NHPP es factible un acercamiento analítico que facilita el proceso de optimización tal como se describe en la mayoría de textos que discuten el mencionado criterio (ver Pham (2007, p.81) para una ilustración de esta idea.) Para el caso del Modelo Goel-Okumoto, hemos señalado la necesidad de resolver el sistema conformado por las ecuaciones 2.3.4 y 2.3.5, donde eliminando el parámetro "$a$" se tiene la ecuación en "$b$":

$$\frac{N(t_n)t_n e^{-bt_n}}{1-e^{-bt_n}} + \sum_{i=1}^{n} \frac{X_i(t_i e^{-bt_i} - t_{i-1}e^{-bt_{i-1}})}{e^{-bt_i} - e^{-bt_{i-1}}} = 0$$

que en la práctica debe resolverse mediante métodos numéricos, preferentemente mediante algoritmos robustos para expresiones no lineales. Esto fue implementado en Octave mediante la función `fzero()`[11] en una versión inicial de nuestras pruebas; sin embargo, pronto constatamos que computacionalmente era igual de efectivo emplear la optimización directa de la función (logarítmica) de verosimilitud que viene dada por la ecuación 2.3.3, al tiempo que reducía la necesidad de codificar los sistemas de ecuaciones derivados como el señalado líneas arriba. En ese sentido, empleamos la función `sqp()` de Octave, correspondiente

[11] La documentación de Octave señala el empleo del algoritmo descrito en Alefeld, Potra y Shi (1995), que incorpora diversas optimizaciones a los métodos numéricos clásicos tales como el de la secante y la interpolación de polinomios.

al algoritmo de Programación Cuadrática Secuencial descrito en la sección 2.6.8.

A continuación presentamos las funciones logarítmicas de verosimilitud a maximizar para los modelos Goel-Okumoto, LLD, ELD y ELI son respectivamente, basadas en la ecuación 2.3.3.

#### 4.3.2.1. Modelo Goel-Okumoto

A partir de la ecuación 1.2.2 se obtiene la expresión reemplazando directamente en 2.3.3:

$$L_{G-O}(a,b) = \sum_{i=1}^{n} X_i \log[a(exp(-bt_{i-1}) - exp(-bt_i))]$$

$$+a(exp(-bt_n) - exp(-bt_0)) - \sum_{i=1}^{n} \log(X_i!)$$

El proceso de optimización intenta maximizar esta expresión mediante el ajuste de los parámetros $a$ y $b$ (ambos positivos), a partir de las muestras correspondientes a los instantes temporales de los eventos de pico $t_i$ y el "conteo" asociado a éstos en $X_i$. De acuerdo con Amrit L Goel y Okumoto (1979), $a$ representa el número experado de errores de software a ser eventualmente detectados, y $b$ es una constante de proporcionalidad.

#### 4.3.2.2. Modelo LLD

A partir de la ecuación 4.2.15 reemplazando en 2.3.3:

$$L_{LLD}(a,b,c,d) = \sum_{i=1}^{n} X_i \log[b \log(\frac{1+ct_i}{1+ct_{i-1}}) + d(t_i - t_{i-1})]$$

$$-b \log(\frac{1+ct_n}{1+ct_0}) - d(t_n - t_0) - \sum_{i=1}^{n} \log(X_i!)$$

Debe observarse la desaparición del parámetro "$a$" en esta expresión; esto significa que el valor máximo de $L_{LLD}$ se da independientemente de aquel parámetro. Por tanto, "$a$" requiere de un criterio ulterior para fijarlo, para lo cual empleamos una minimización complementaria basada en la métrica SSE. Esta idea también es aplicable a los modelos subsiguientes.

#### 4.3.2.3. Modelo ELD

A partir de la ecuación 4.2.16 reemplazando en 2.3.3:

$$L_{ELD}(a,b,c,d) = \sum_{i=1}^{n} X_i \log[-b(e^{-ct_i} - e^{-ct_{i-1}}) + d(t_i - t_{i-1})]$$

$$+b(e^{-ct_n} - e^{-ct_0}) - d(t_n - t_0) - \sum_{i=1}^{n} \log(X_i!)$$

#### 4.3.2.4. Modelo ELI

A partir de la solución para el modelo de tendencia ideal 4.2.3 reemplazando en 2.3.3:

$$L_{ELI}(a,k,D_0,N_0) = \sum_{i=1}^{n} X_i \log[ELI^{-1}_{a,k,D_0,N_0}(t_i) - ELI^{-1}_{a,k,D_0,N_0}(t_{i-1})]$$

$$-[ELI^{-1}_{a,k,D_0,N_0}(t_n) - ELI^{-1}_{a,k,D_0,N_0}(t_0)] - \sum_{i=1}^{n} \log(X_i!)$$

donde $ELI^{-1}$ es la función inversa de $ELI$ definida por la ecuación 4.2.4:

$$ELI_{a,k,D_0,N_0}(N) = \frac{1}{k}\{(N - N_0) + \frac{(1 - D_0)[\exp(-a[N - N_0]) - 1]}{a}\}$$

### 4.3.3. Contraste Empírico de Modelos para Tendencia Estocástica

#### 4.3.3.1. Métricas de Bondad de la estimación paramétrica

Emplearemos las métricas NRMSE y AIC para la comparación de los modelos (ver el anexo "Medidas de Capacidad Predictiva" para las respectivas definiciones.)

La métrica NRMSE ha sido empleada al analizar el ajuste de los modelos para tendencia ideal, por lo que es interesante contar con su versión estocástica; por otra parte, la métrica AIC ha sido presentada en 2.3.2.; en ambos casos un mejor ajuste corresponde a un valor numéricamente menor.

#### 4.3.3.2. Resultados de la estimación paramétrica

La tabla 4.10 contiene el número de picos detectados para los días considerados en el análisis. Dicha tabla muestra la tasa porcentual de picos en relación a los días en su última columna.

| Proy. | #Picos | #Dias | Tasa % |
|---|---|---|---|
| SP | 118 | 401 | 29.43 |
| RE | 32 | 88 | 36.36 |
| AD | 19 | 108 | 17.59 |
| AS | 13 | 108 | 12.04 |
| JP | 9 | 41 | 21.95 |
| CT | 24 | 98 | 24.49 |
| PO | 17 | 55 | 30.91 |

Cuadro 4.10: Picos detectados para tendencia estocástica

| Proy. | LLD | ELI | ELD | GOEL |
|---|---|---|---|---|
| SP | 4.12 | 8.21 | 4.54 | 4.62 |
| RE | 7.05 | 7.36 | 6.27 | 10.45 |
| AD | 5.17 | 5.17 | 5.19 | 5.19 |
| AS | 8.21 | 8.27 | 8.14 | 8.15 |
| JP | 9.31 | 8.21 | 9.03 | 9.33 |
| CT | 5.72 | 8.33 | 4.55 | 4.43 |
| PO | 8.67 | 9.55 | 8.40 | 8.40 |

Cuadro 4.11: NRMSE para tendencia estocástica.

La tabla 4.11 contiene el valor NRMSE obtenido para la tendencia estocástica de los modelos que hemos propuesto. Observar que no hay un claro modelo vencedor.

En cuanto a la métrica AIC, los resultados se proporcionan en la tabla 4.12. El resultado tampoco es concluyente de acuerdo a los criterios de diferencia de AIC referidos en la sección 2.3.2: dadas las diferencias obtenidas, debemos concluir que cada modelo resulta ser superior a los otros en al menos uno de los proyectos, lo que apunta a la ocurrencia de distintas dinámicas estocásticas durante el desarrollo de éstos. Por tanto, ninguno de los modelos considerados puede ser considerado adecuado como explicación general para este componente de los datos; futuras propuestas en esta dirección pueden emplear los modelos propuestos como referencia para contrastar, siguiendo el procedimiento aquí descrito.

#### 4.3.3.3. Resultados para Intensidad de Fallos

La tasa instantánea de ocurrencia de eventos (conocida como intensidad de fallos en la teoría de fiabilidad) proporciona una estimación del grado de "inestabilidad" de la dinámica del desarrollo para un momento determinado. Esta inestabilidad tiene diversas causas, algunas de las cuales se han reseñado en la

| Proy. | LLD | ELI | ELD | GOEL |
|---|---|---|---|---|
| SP | 667.84 | 668.11 | 660.36 | 649.54 |
| RE | 166.02 | 162.00 | 171.24 | 167.94 |
| AD | 125.63 | 121.61 | 125.45 | 121.55 |
| AS | 82.34 | 77.35 | 82.05 | 78.24 |
| JP | 45.69 | 41.02 | 46.64 | 41.67 |
| CT | 134.38 | 131.26 | 133.74 | 130.15 |
| PO | 81.79 | 77.31 | 81.15 | 77.15 |

Cuadro 4.12: AIC para tendencia estocástica.

| Proy. | LLD | ELI | ELD | GOEL |
|---|---|---|---|---|
| SP | 2.36 | 2.86 | 2.35 | 2.35 |
| RE | 3.17 | 3.17 | 3.08 | 2.08 |
| AD | 0.92 | 0.91 | 0.90 | 0.90 |
| AS | 0.61 | 0.59 | 0.58 | 0.58 |
| JP | 0.76 | 0.71 | 0.76 | 0.76 |
| CT | 2.10 | 2.04 | 2.04 | 1.86 |
| PO | 1.83 | 1.69 | 1.77 | 1.77 |

Cuadro 4.13: Inestabilidad al inicio del proyecto

sección 1.2.2. Desde el punto de vista de nuestros modelos, la intensidad se obtiene directamente a partir de su definición (ecuación 2.4.1) efectuando una derivación de los valores ajustados. El resultado se muestra en las tablas 2.4.1 y 4.14 para los instantes iniciales y finales de los proyectos, respectivamente.

Lo primero que salta a la vista es la homogeneidad en los resultados obtenidos por los modelos considerando los proyectos individualmente. Como es de esperarse, la inestabilidad tiende a disminuir a lo largo del desarrollo; sin embargo, cada proyecto presenta valores inherentes que consideramos dependen fundamentalmente del contexto humano en el que despliegan. Así por ejemplo, el proyecto "RE" estuvo dirigido a usuarios que carecían de tiempo para participar del análisis, lo que exigió diversas codificaciones "tentativas" (y sus respectivos retrocesos) a fin de que -en su escaso tiempo disponible- los usuarios se limiten a "validar" las ideas que se estaban implementando.

Totalmente distinto es el caso del proyecto AD, el cual involucró un desarrollo muy predecible casi en su totalidad, si bien fue ocasionalmente interrumpido a la espera de ciertas definiciones complementarias a cargo del usuario final.

| Proy. | LLD | ELI | ELD | GOEL |
|---|---|---|---|---|
| SP | 0.69 | 0.70 | 0.43 | 0.43 |
| RE | 1.03 | 1.06 | 1.06 | 0.87 |
| AD | 0.64 | 0.64 | 0.62 | 0.62 |
| AS | 0.28 | 0.30 | 0.27 | 0.27 |
| JP | 0.76 | 0.70 | 0.76 | 0.75 |
| CT | 0.42 | 0.54 | 0.38 | 0.31 |
| PO | 0.84 | 0.94 | 0.76 | 0.76 |

Cuadro 4.14: Inestabilidad al término del proyecto

#### 4.3.3.4. Relación Tamaño-Picos

Siendo los proyectos de similares características, es razonable que exista una fuerte correlación entre el tamaño final del proyecto en líneas de código $N(t_F)$ y el acumulado de eventos estocásticos $M(t_F)$ con valor medio $m(t_F)$. Al respecto, postulamos el modelo lineal:

$$\widehat{m(t_F)} \approx \beta_1 + \beta_2 N(t_F) \tag{4.3.2}$$

con los tamaños disponibles para nuestros proyectos, esto produce el modelo ajustado con coeficiente de determinación $R^2 = 0{,}976$:

$$\widehat{m(t_F)} = 3{,}867 + 0{,}00719 N(t_F)$$

#### 4.3.3.5. Predicción vía tendencia estocástica

Dado un conjunto de muestras correspondientes a los primeros días de ejecución del proyecto, es posible hacer un ajuste del modelo de tendencia estocástica de acuerdo a lo explicado en la sección 4.3.2. Este ajuste contiene los parámetros que definen completamente la función de valor esperado de fallos $m(t)$ y permite establecer la secuencia de valores:

$$\Theta = \{m(0), m(1), ..., m(i), ...\}$$

Esta secuencia es estrictamente creciente para el modelo ELI (ecuación 4.2.6.) Disponiendo de una aproximación para el total de eventos estocásticos $\widehat{m(t_F)}$ (para lo cual se puede emplear el modelo denotado por la ecuación 4.3.2) es posible obtener el mínimo valor de $i$ tal que:

$$m(i) \geq \widehat{m(t_F)}$$

A fin de verificar este argumento consideramos realizar el ajuste de los datos correspondientes a los primeros 60 días (efectivos) de avance. La tabla 4.15

**Algoritmo 4.2** Predicción vía tendencia estocástica

```
Entradas:
TLOC: Tamaño final del código fuente
LOC: Vector con conteo de picos de primeros días de proyecto
DLIM: Limite admisible de días
BETA: Parámetros de ajuste de total de eventos

Salida: Duración tentativa de proyecto, o error.

Funciones auxliares:
FIT_ELI(LOC): Ajuste de parámetros de modelo ELI
ELI(PARAM,DIA): Predicción LOC del modelo ELI
LEN(LOC): Longitud de vector

PASO 1: Ajustar parámetros modelo ELI a partir data LOC
  PARAMS <- FIT_ELI(LOC)
  EV.TOTAL <- BETA1+BETA2*TLOC
  DIA <- LEN(LOC)

PASO 2: Obtener tamaño según ELI
  CURRENT <- ELI(PARAMS,DIA)

PASO 3: Validar si se alcanza el tamaño
  Si CURRENT >= EV.TOTAL : mostrar dia y terminar
  Si DIA >= DLIM: terminar con error
  DIA <- DIA + 1
  Volver a PASO 2
```

muestra (en su segunda columna) las predicciones temporales generadas por este método utilizando los valores exactos para el tamaño final del código fuente (tabla 3.4.) El algoritmo 4.2 implementa esta idea.

## 4.4. Sobre la Consolidación de los Modelos

Las dos últimas secciones nos proporcionan sendos procedimientos predictivos para el tiempo de desarrollo (e indirectamente de esfuerzo) de nuestro conjunto

| Proy. | Pred. | Real |
|---|---|---|
| SP | 494 | 401 |
| RE | 57 | 88 |
| AD | 65 | 108 |
| AS | 41 | 108 |
| JP | 24 | 41 |
| CT | 68 | 98 |
| PO | 100 | 55 |

Cuadro 4.15: Duración de proyectos (en días) según predicción de tendencia estocástica.

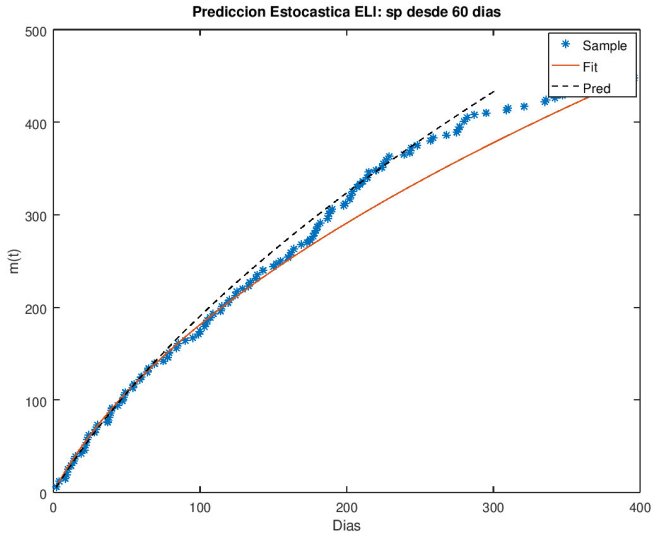



Figura 4.3.1: Predicción de tendencia estocástica proyecto SP

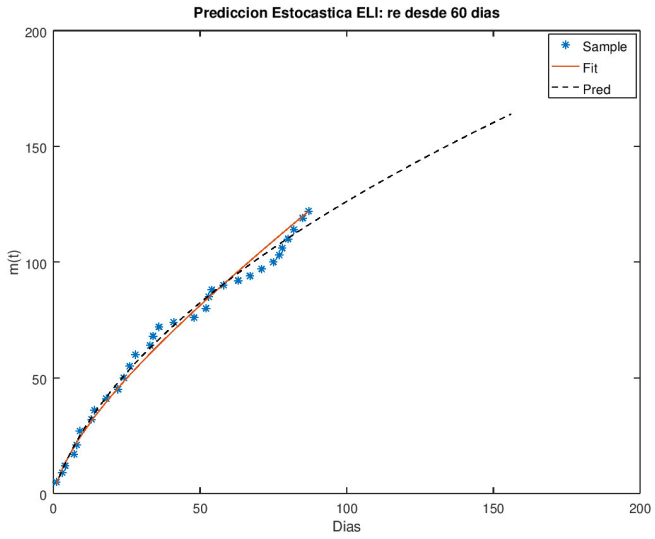



Figura 4.3.2: Predicción de tendencia estocástica proyecto RE

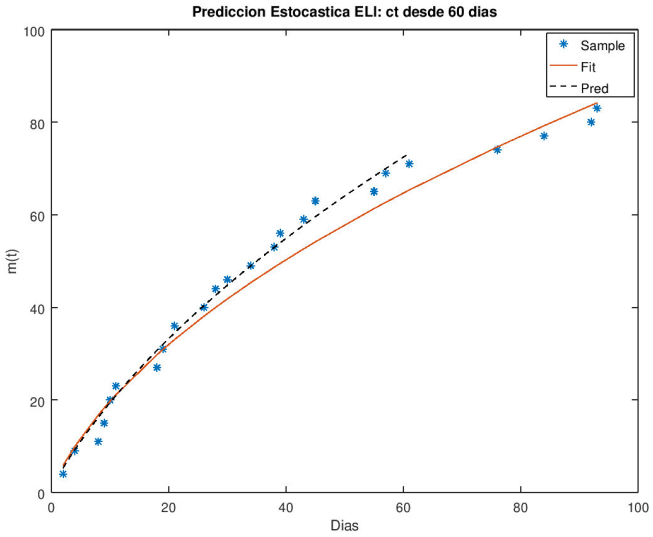



Figura 4.3.3: Predicción de tendencia estocástica proyecto CT

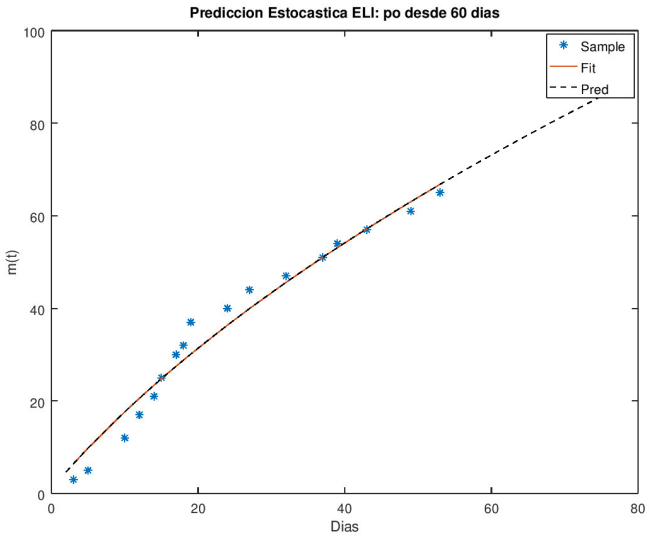



Figura 4.3.4: Predicción de tendencia estocástica proyecto PO

| Proy. | Pred.Dias | Pred.Sem. | Real Dias | Real Sem. |
|---|---|---|---|---|
| SP | 494 | 72 | 401 | 58 |
| RE | 57 | 9 | 88 | 14 |
| AD | 65 | 10 | 108 | 16 |
| AS | 41 | 7 | 108 | 16 |
| JP | 24 | 4 | 41 | 7 |
| CT | 68 | 11 | 98 | 15 |
| PO | 100 | 15 | 55 | 9 |

Cuadro 4.16: Duración de proyectos (en días y semanas) según predicción de tendencia estocástica.

de proyectos de muestra, el cual se ha derivado de diversas consideraciones teóricas y empíricas. Al respecto, podemos intentar el planteamiento de un modelo consolidado de predicción temporal que proporcione el mejor ajuste a nuestros datos.

La tabla 4.16 es una adaptación de 4.15 donde se han introducido los valores correspondientes en semanas, a fin de homogeneizar la escala con el método de tendencia ideal. A modo de ejemplo, considérese los 401 días del proyecto "SP", los cuales al ser divididos por siete arrojan (tras redondear) 57 semanas. En tanto la tendencia estocástica se ha basado en el promedio móvil a tres días, en la práctica se han "descartado" dos días al inicio y dos al final del rango considerado; por tanto, es razonable agregar una semana al valor antes obtenido con lo que obtenemos las 58 semanas, tal como se señala en la tabla 4.8 para la tendencia ideal. Este mismo procedimiento se ha aplicado para convertir la columna de valores correspondientes a la "predicción estocástica" a escala semanal.

Consolidando las tablas 4.8 (columna para 4 semanas de datos iniciales) y 4.16 en 4.17, obtenemos las predicciones de los dos métodos antes descritos. Asimismo, hemos agregado los errores porcentuales en los que incurren estos métodos. Numéricamente, una predicción que "combina" ambas tendencias podría potencialmente generar un error más reducido que cualquiera de las anteriores por separado; en particular, hemos considerado combinaciones lineales de los resultados de estas tendencias, denominando $I$ a la predicción de tendencia ideal y $E$ a la predicción de tendencia estocástica:

$$Pred_{total} = aI + bE$$

donde $a$ y $b$ son coeficientes a determinarse a fin de minimizar el error total.

Para la expresión de "error total" usualmente se considera la suma de los cuadrados de los errores individuales, lo que convierte al cálculo de los coeficientes en un problema similar al descrito en la sección 2.1.3:

| Proyecto | Ideal | Estoc. | Real | Err %Ideal | Err %Estoc. |
|---|---|---|---|---|---|
| SP | 16 | 72 | 58 | -72.41 | 24.14 |
| RE | 6 | 9 | 14 | -57.14 | -35.71 |
| AD | 14 | 10 | 16 | -12.5 | -37.5 |
| AS | 40 | 7 | 16 | 150 | -56.25 |
| JP | 5 | 4 | 7 | -28.57 | -42.86 |
| CT | 7 | 11 | 15 | -53.33 | -26.67 |
| PO | 5 | 15 | 9 | -44.44 | 66.67 |

Cuadro 4.17: Predicciones para duración de proyectos y errores, según tendencias ideal y estocástica.

$$Err = \sum_{i=1}^{NP}(1 - aI_i - bE_i)^2 \rightarrow$$

$$\frac{\partial Err}{\partial a} = \sum_{i=1}^{NP} 2(1 - aI_i - bE_i)(-I_i)$$

$$\frac{\partial Err}{\partial b} = \sum_{i=1}^{NP} 2(1 - aI_i - bE_i)(-E_i)$$

Igualando a cero las derivadas se obtiene el sistema de ecuaciones:

$$\begin{aligned} (\sum I_i^2)a + (\sum I_i E_i)b &= \sum I_i \\ (\sum I_i E_i)a + (\sum E_i^2)b &= \sum E_i \end{aligned}$$

La solución al mismo se implementa fácilmente en Octave, donde mostramos la solución para las incógnitas de acuerdo a los datos de la tabla 4.17:

```
octave> type mincon
% minimizacion consolidada
function mincon(I,E,R)
I=I./R; E=E./R;
sumi=sum(I); sume=sum(E);
sumi2=sum(I.^2);
sume2=sum(I.^2);
sumie=sum(I.*E);
X=[sumi2,sumie;sumie,sume2];
Y=[sumi;sume];
X\Y
endfunction
octave> mincon([16;6;14;40;5;7;5],[72;9;10;7;4;11;15],[58;14;16;16;7;15;9]);
ans =
   0.46739
   0.49074
```

Lo anterior significa que nuestra fórmula de predicción consolidada (redondeada a un decimal) es:

| Proy. | Pred.Cons. | Real | Err % |
|---|---|---|---|
| SP | 42.8 | 58 | -26.21 |
| RE | 7.23 | 14 | -48.36 |
| AD | 11.48 | 16 | -28.25 |
| AS | 22.23 | 16 | 38.94 |
| JP | 4.31 | 7 | -38.43 |
| CT | 8.68 | 15 | -42.13 |
| PO | 9.7 | 9 | 7.78 |

Cuadro 4.18: Duración de proyectos: predicciones consolidadas (en semanas) mediante fórmula lineal, y error relativo.

$$Pred_{total} = 0{,}47I + 0{,}49E$$

La tabla 4.18 muestra las predicciones consolidadas y su error relativo. Si se compara con la tabla 4.17 se puede apreciar que el porcentaje de error sigue siendo considerable pero (al menos) se han eliminado los valores extremos. En nuestra opinión este mecanismo de consolidación no aporta una ventaja considerable por lo que se debe estudiar otras formulaciones alternativas a la combinación lineal, y mientras tanto mantener los resultados de ambos métodos como recomendaciones independientes para los gestores de los proyectos[12].

[12]Otro motivo para no recomendar esta formulación lineal radica en la inestabilidad de los coeficientes $a$ y $b$ obtenidos para otros conjuntos de valores de la predicción de tendencia ideal (6 y 8 semanas.)

# Conclusiones

## La Problemática y la Propuesta de Solución

El presente estudio ha constado de dos enfoques complementarios, a saber, la formulación matemática de un modelo teórico y su aplicación a la información empírica que disponemos.

Si bien estos enfoques han sido expuestos secuencialmente en el presente documento, durante la investigación ambos han interactuado e influido mutuamente; por ejemplo, las hipótesis teóricas cuya justificación se basó en propuestas relacionadas a la noción de complejidad de software, en realidad fueron sugeridas como resultado de la observación visual de los gráficos de avance consolidado de la codificación de los proyectos. Del mismo modo, las limitaciones del modelo analítico sugirieron el planteamiento de una versión estocástica que explicase la "irregularidad" de los mencionados patrones exhibidos por los datos. Asimismo, la información histórica de tipo administrativo sugirió la existencia de inestabilidad en la actividad de codificación reflejada en tendencias temporalmente "erráticas".

Tal como señalamos al inicio de este documento, el problema que estamos abordando es relativamente novedoso pero está fuertemente emparentado con el clásico problema de la estimación de esfuerzo. Consideramos que la propuesta realizada, si bien no intenta proporcionar una respuesta directa, sí puede ser considerada una herramienta de gestión auxiliar para el control de proyectos en curso.

## Beneficios que aporta la propuesta

El control de los costos en los que incurren los proyectos es fundamental para la gestión exitosa de los mismos. Esto se implementa de forma preliminar mediante una adecuada planificación de las actividades (en el caso del software, con la estimación del esfuerzo), pero también es imperativo hacerlo a lo largo de la ejecución de las tareas involucradas. Para esto existen diversas herramientas de gestión, pero como comentamos al inicio, éstas no suelen ser convenientes para una actividad en la que los requerimientos son cambiantes. Es plausible afirmar

que muchos de los grandes fracasos en los proyectos de software no solo se deben a un error de planificación, sino a un control inadecuado de los mencionados cambios inesperados. A tal efecto, esta propuesta proporciona dos modelos de la dinámica del desarrollo basado en variables fácilmente mesurables (el código fuente que va siendo generado), así como su extrapolación temporal; esto permite al gestor sincerar los tiempos totales de las actividades de desarrollo en forma anticipada, constituyéndose en una herramienta de control de gran utilidad.

De otro lado, desde un punto de vista estrictamente académico, consideramos que la discusión de la sección 1.2.2 puede ser de utilidad a futuros investigadores interesados en el problema general de la estimación del esfuerzo en proyectos de software. Esta sección pone énfasis en las dificultades que descubrimos al principio de este trabajo, así como en diversos cuestionamientos señalados por profesionales muy experimentados; si bien el tono general es muy negativo, la contribución radica en advertir sobre muchas aproximaciones que han fracasado o que han llegado a callejones sin salida.

## Implementación de la propuesta

La implementación de esta propuesta requiere de facilidades de cómputo básicas así como familiaridad en la implementación de algoritmos matemáticos usuales (por ejemplo, para aproximar la solución de ecuaciones diferenciales.) En nuestro caso todo esto fue ejecutado con ayuda del software Octave, pero cualquier programa similar puede cumplir este propósito.

En la práctica, tal vez lo más difícil sea disponer de acceso al repositorio de cambios en el código fuente dadas las usuales consideraciones de propiedad intelectual y seguridad en general; sin embargo, como se explica en la sección 3.3, es factible realizar el procesamiento sin necesidad de disponer del código fuente como tal, en tanto sólo se utilizan los conteos de línea respectivos.

El interesado puede consultar los listados de los programas principales que se desarrollaron para este trabajo en los anexos, con lo que es relativamente sencillo construir un aplicativo que automatice el procesamiento de los datos con los algoritmos descritos en este documento.

## Propuestas para investigaciones complementarias

En nuestra opinión, la limitación más evidente con la que nos hemos encontrado en este trabajo corresponde al número de proyectos considerados. Si bien disponemos de información abundante para cada uno de éstos, es claro que siete proyectos resultan insuficientes para justificar y pretender extrapolar resultados,

especialmente cuando la magnitud de los errores reportados es tan elevada en comparación con los que se suelen obtener en los cálculos de la ingeniería y/o los obtenidos en experimentos científicos exitosos.

A tal efecto consideramos necesario disponer de una base mayor de proyectos. Asimismo, anteriormente hemos mencionado que los proyectos considerados han estado dirigidos a satisfacer las necesidades de dos empresas; en ese sentido, resulta razonable efectuar el análisis consolidado de los proyectos de cada empresa, validando la efectividad de modelos para cada una de éstas. Nuevamente, esto requiere de una muestra razonablemente grande por cada empresa considerada.

Líneas arriba comentábamos la dificultad de extraer datos significativos de proyectos reales por criterios relacionados con la confidencialidad de la información. Al respecto, hemos utilizado el clásico "conteo de líneas" como métrica que mitiga estos inconvenientes; sin embargo, sabemos que ésta es una muy gruesa aproximación a los programas y su complejidad. En ese orden de ideas resulta razonable proponer el uso de métricas que (respetando la confidencialidad de la fuente) expresan más información del software; verbigracia, el conteo de unidades léxicas y/o sintácticas, así como ponderaciones según el tipo de componente (tal como propone el análisis de puntos de función.)

De otro lado, disponiéndose de la información que describe un conjunto mayor de proyectos, la consolidación de las tendencias que hemos propuesto podría ser sometida a un análisis más fino tal como el que indica la estadística bayesiana. Por ejemplo, la tendencia ideal podría considerarse como una opinión preliminar (prior) a ser complementada por la tendencia estocástica, generándose una tendencia consolidada (posterior.)

Como se indicó, este trabajo se ha restringido a modelos matemáticos y estadísticos tradicionales en el sentido de no incluirse técnicas de inteligencia artificial (IA.) En nuestra opinión el uso de la IA para el problema de la estimación de esfuerzo ha generado una gran cantidad de resultados insustanciales originados a partir de datos reconocidamente dudosos y cuantitativamente insignificantes (cfr. la sección 1.2.2), los que son procesados "a ciegas" a través de software de "machine learning". Con esta advertencia, consideramos que las técnicas de IA sí pueden proporcionar algunas ventajas sobre el modelamiento tradicional para la dinámica del software: nuestras restricciones para los proyectos considerados (especialmente referentes al equipo de desarrolladores) son claramente insuficientes de cara a los múltiples escenarios y modalidades en que se efectúan los proyectos de software; así, resulta por de más dudoso esperar que un único modelo matemático (como el aquí propuesto) tenga validez universal. Las técnicas de IA precisamente pueden emplearse para catalogar los proyectos según su dinámica a fin de establecer ulteriormente sendos modelos matemáticos.

Para terminar, el factor humano fue mencionado pero explícitamente excluido. Según la experiencia del autor, la "mente" de cada desarrollador incide notablemente en sus respectivos patrones de productividad; así, es de esperarse que la dinámica de codificación de los desarrolladores (individualmente considerada) tenga características distintivas asociadas a la tipología intelectual de estos profesionales.

# Bibliografía


Abran, Alain (2015). *Software project estimation: The fundamentals for providing high quality information to decision makers*. John Wiley & Sons.

Akaike, Hirotugu (1974). "A new look at the statistical model identification". En: *IEEE transactions on automatic control* 19.6, págs. 716-723.

Albrecht, Allan J (1979). "Measuring application development productivity". En: *Proc. Joint Share, Guide, and IBM Application Development Symposium, 1979*.

Alefeld, GE, Florian A Potra y Yixun Shi (1995). "Algorithm 748: enclosing zeros of continuous functions". En: *ACM Transactions on Mathematical Software (TOMS)* 21.3, págs. 327-344.

Aman, Hirohisa y Takahiro Ohkochi (2010). "An application of growth curve model for predicting code churn in open source development". En: *Proceedings of the 9th Joint Conference on Knowledge-Based Software Engineering, JCKBSE 2010*.

Anderson, D y K Burnham (2002). *Model selection and multi-model inference*. 2da. ed. Springer-Verlag.

Apostol, Tom M (1969). *Calculus, Volume II: Multi-Variable Calculus and Linear Algebra, with Applications to Differential Equations and Probability*. John Wiley & Sons.

Arnold, K. y J. Gosling (1997). *El lenguaje de programación Java*. Addison-Wesley Iberoamericana/Domo Editores. Wilmington, Delaware.

Arnuphaptrairong, Tharwon (2018). "The State of Practice of Software Cost Estimation: Evidence From Thai Software Firms". En: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 2.

Beedle, Mike (2001). *Manifesto for Agile Software Development*. `https://agilemanifesto.org/`. Accedido el 10/2/2021.

Belady, Laszlo A. y Meir M Lehman (1976). "A model of large program development". En: *IBM Systems journal* 15.3, págs. 225-252.

Boehm, B. W. (1981). *Software engineering economics*. Prentice-hall.

Boehm, B. W., R. Madachy y B. Steece (2000). *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR.

Boehm, Barry W y Ricardo Valerdi (2008). "Achievements and challenges in cocomo-based software resource estimation". En: *IEEE software* 5, págs. 74-83.

Bramble, Michael y col. (2003). *Handbook for Software Cost Estimation.*

Briand, L. C., V. R. Basili y W. M. Thomas (1993). "Recognizing patterns for software development prediction and evaluation". En: *In Analytical Methods in Software Engineering Economics (pp. 151-170). Springer, Berlin, Heidelberg.*

Brooks Jr, Frederick P (1995). *The mythical man-month: essays on software engineering.* Pearson Education.

Brown, Kyle y col. (2003). *Enterprise java programming with ibm websphere.* Addison-Wesley Professional.

Burden, R. L., Faires y J. D (2010). *Numerical analysis.* 9na ed. Cengage Learning.

Cant, SN, D Ross Jeffery y Brian Henderson-Sellers (1995). "A conceptual model of cognitive complexity of elements of the programming process". En: *Information and Software Technology* 37.7, págs. 351-362.

Čeke, D., Milašinović y B (2015). "Early effort estimation in web application development". En: *Journal of Systems and Software, 103, 219-237.*

Charette, R. N. (2005). "Why software fails [software failure]". En: *Ieee Spectrum, 42(9), 42-49.*

Chemuturi, Murali (2009). *Software estimation best practices, tools & techniques: a complete guide for software project estimators.* J. Ross Publishing.

Chhabra, J. K., K. K. Aggarwal e Y. Singh (2003). "Code and data spatial complexity: Two important software understandability measures". En: *Information and software Technology, 45(8), 539-546.*

Cohn, M. (2005). *Agile estimating and planning.* Pearson Education.

Corder, Gregory W y Dale I Foreman (2014). *Nonparametric statistics: A step-by-step approach.* 2da. ed. John Wiley & Sons.

Cosmic (2015). *The COSMIC Functional Size Measurement Method (Measurement Manual).* Common Software Measurement International Consortium. Version 4.0.1.

Coulter, N. S. (1983). "Software science and cognitive psychology". En: *IEEE Transactions on Software Engineering, (2), 166-171.*

Ding, Jie, Vahid Tarokh y Yuhong Yang (2018). "Model selection techniques: An overview". En: *IEEE Signal Processing Magazine* 35.6, págs. 16-34.

Eaton, John W. (2018). *About GNU Octave.* `http://www.gnu.org/software/octave/about.html`. Accedido el 15/1/2019.

Emam, K. El, Koru y A. G (2008). "A replicated survey of IT software project failures". En: *IEEE software, (5), 84-90.*

Fisher, Irving (1999). "La teoría de la deuda-deflación en las grandes depresiones". En: *Problemas del Desarrollo* 30.119, págs. 189-210.

Folgieri, Raffaella y col. (2011). "A revised Web Objects method to estimate Web application development effort". En: *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*. ACM, págs. 59-64.

Foulds, Leslie R (1981). *Optimization techniques: an introduction*. Springer.

Galorath, Daniel D y Michael W Evans (2006). *Software sizing, estimation, and risk management: when performance is measured performance improves*. Auerbach Publications.

Gamma, E. y col. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Ghorpade, Sudhir R y Balmohan V Limaye (2008). "A course in calculus and real analysis". En: *The Australian Mathematical Society*, pág. 211.

Glass, Robert L (2004). "Some heresy regarding software engineering". En: *IEEE Software* 4, págs. 104-102.

Goel, Amrit L y Kazu Okumoto (1979). "Time-dependent error-detection rate model for software reliability and other performance measures". En: *IEEE transactions on Reliability* 28.3, págs. 206-211.

Goel, Amrit L. (1985). "Software reliability models: Assumptions, limitations, and applicability". En: *IEEE Transactions on software engineering* 12, págs. 1411-1423.

Gómez, R. (2005). *Programación Avanzada en Shell*. `www.informatica.us.es/~ramon/articulos/Programacion-BASH.pdf`. Accedido el 23/12/2018.

Gottesdiener, Ellen (1995). "RAD realities: Beyond the hype to how RAD really works". En: *Application Development Trends* 2.8, págs. 28-38.

Gujarati, Damodar y Dawn Porter (2010). *Econometría*. 5ta. ed. McGraw-Hill.

Halstead, Maurice Howard y col. (1977). *Elements of software science*. Vol. 7. Elsevier New York.

Harrison, Warren y Curtis Cook (1987). "A micro/macro measure of software complexity". En: *Journal of Systems and Software* 7.3, págs. 213-219.

Heemstra, Fred J (1992). "Software cost estimation". En: *Information and software technology* 34.10, págs. 627-639.

Hetzel, William C (1995). "The sorry state of software practice measurement and evaluation". En: *Journal of Systems and Software* 31.2, págs. 171-179.

Huang, Chin-Yu, Sy-Yen Kuo y Michael R Lyu (2007). "An assessment of testing-effort dependent software reliability growth models". En: *IEEE transactions on Reliability* 56.2, págs. 198-211.

IFPUG (2010). "Function Point Counting Practices Manual". En: *Release 4.3.1*.

ISBSG (n.d.). *International Software Benchmarking Standards Group*. `http://isbsg.org/`. Accedido el 13/01/2019.

ISO (abr. de 2005). *ISO/IEC 2382:2015 Information technology – Vocabulary*. Standard. Geneva, CH: International Organization for Standardization.

Jeffery, Ross, Melanie Ruhe e Isabella Wieczorek (2000). "A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data". En: *Information and software technology* 42.14, págs. 1009-1016.

Jones, C. (2011). "Sources of Software Benchmarks". En: *Accedido el 18 de Mayo de 2018.* URL: `https://www.namcook.com/Articles/SourcesOfSoftwareBenchmarks19.doc`.

Jones, Capers (2004). "Software project management practices: Failure versus success". En: *CrossTalk: The Journal of Defense Software Engineering* 17.10, págs. 5-9.

— (2008). *Applied software measurement: global analysis of productivity and quality*. McGraw-Hill Education Group.

— (2013). *The technical and social history of software engineering*. Addison-Wesley.

— (2017). *A Guide to Selecting Software Measures and Metrics*. CRC Press.

Jørgensen, Magne (2007). "A critique of how we measure and interpret the accuracy of software development effort estimation". En: *First International Workshop on Software Productivity Analysis and Cost Estimation. Information Processing Society of Japan, Nagoya*. Citeseer.

Jureta, Ivan (2015). *The Design of Requirements Modelling Languages*. Springer.

Keshta, I. M. (2017). "Software Cost Estimation Approaches: A Survey". En: *Journal of Software Engineering and Applications, 10(10), 824.*

Kitchenham, B., Taylor y N. R (1984). "Software cost models". En: *ICL technical journal, 4(1), 73-102.*

Kitchenham, B. A. (1990). "Measuring to manage". En: *Managing complexity in software engineering* 17.

Kitchenham, Barbara A y col. (2001). "What accuracy statistics really measure [software estimation]". En: *IEE Proceedings-Software* 148.3, págs. 81-85.

Laird, Linda M y M Carol Brennan (2006). *Software measurement and estimation: a practical approach*. John Wiley & Sons.

Lancaster, Peter y Kestutis Salkauskas (1986). *Curve and surface fitting: an introduction*. Academic press.

Lederer, Albert L y col. (1990). "Information system cost estimating: a management perspective". En: *MIS Quarterly*, págs. 159-176.

Lehman, M. M. (1980). "Programs, life cycles, and laws of software evolution". En: *Proceedings of the IEEE, 68(9), 1060-1076.*

Lokan, C. J. (2005). "Function points". En: *Advances in Computers, 65, 297-347.*

Longley, D., Shain y M. Eds (1985). "Macmillan dictionary of information technology". En: *Springer*.

Lyu, Michael R y col. (1996). *Handbook of software reliability engineering*. Vol. 222. IEEE computer society press CA.

Matson, Jack E, Bruce E Barrett y Joseph M Mellichamp (1994). "Software development cost estimation using function points". En: *IEEE Transactions on Software Engineering* 20.4, págs. 275-287.

McCabe, Thomas J (1976). "A complexity measure". En: *IEEE Transactions on software Engineering* 4, págs. 308-320.

McConnell, Steve (2006). *Software estimation: demystifying the black art*. Microsoft press.

Mendes, Emilia (2007). "Cost estimation techniques for web projects". En: *IGI Global*.

— (2014). *Practitioner's Knowledge Representation*. Springer.

Mendes, Emilia y Nile Mosley (2006). "Web engineering". En: *Springer Science & Business Media*.

Mens, T. y S. E. Demeyer (2008). "Software Evolution". En: *Springer-Verlag Berlin Heidelberg*.

Mills, Terence C (2011). *The foundations of modern time series analysis*. Springer.

Molokken, Kjetil y Magne Jorgensen (2003). "A review of software surveys on software effort estimation". En: *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*. IEEE, págs. 223-230.

Morris, D. y B. E. Tamm (1993). *Concise Encyclopedia of Software Engineering*. Pergamon Press. ISBN: 978-0080362144.

Nocedal, Jorge y Stephen Wright (2006). *Numerical optimization*. Springer Science & Business Media.

Norden, Peter V (1977). "Project life cycle modeling: Background and application of the life cycle curves". En: *Software Phenomenology - Working papers of the Software Life Cycle Management Workshop*.

Osmanbegović, Edin, Mirza Suljić y Haris Agić (2017). "A review of estimation of software products development costs". En: *Ekonomski vjesnik/Econviews-Review of Contemporary Business, Entrepreneurship and Economic Issues* 30.1.

Paolella, M.S. (2018). *Linear Models and Time-Series Analysis: Regression, ANOVA, ARMA and GARCH*. Wiley Series in Probability and Statistics. Wiley. ISBN: 9781119431909. URL: `https://books.google.com.pe/books?id=vkNxDwAAQBAJ`.

Papoulis, Athanasios y S Unnikrishna Pillai (2002). *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education.

Pham, Hoang (2007). *System software reliability*. Springer Science & Business Media.

PHP Group (2018). *The PHP Manual*. Ed. por Peter Cowburn. `http://php.net/manual/en/intro-whatis.php`. Accedido el 15/1/2019.

Pilato, C Michael, Ben Collins-Sussman y Brian W Fitzpatrick (2008). *Version Control with Subversion: Next Generation Open Source Version Control*. O'Reilly Media, Inc.

Pillai, K., Nair y V. S (1997). "A model for software development effort and cost estimation". En: *IEEE Transactions on Software Engineering, (8), 485-497.*

PMI (2013). "Software Extension to PMBOK Guide". En: *Fifth edition. Project Management Institute.*

Putnam, Lawrence H. (1978). "A general empirical solution to the macro software sizing and estimating problem". En: *IEEE transactions on Software Engineering* 4, págs. 345-361.

Riordan, JS (1977). "An evolution dynamics model of software systems development". En: *Software Phenomenology-Working Papers of the (First) SLCM Workshop, Airlie, Virginia, Aug 1977*, págs. 339-360.

Ross, Sheldon M (2014). *Introduction to probability models*. Academic press.

Rudin, Walter y col. (1976). *Principles of mathematical analysis*. 3ra ed. McGraw-hill New York.

Sharma, Kapil y col. (2010). "Selection of optimal software reliability growth models using a distance based approach". En: *IEEE Transactions on Reliability* 59.2, págs. 266-276.

Shen, Vincent Yun, Samuel D. Conte y Hubert E. Dunsmore (1983). "Software science revisited: A critical analysis of the theory and its empirical support". En: *IEEE Transactions on Software Engineering* 2, págs. 155-165.

Shepperd, Martin, Michelle Cartwright y Gada Kadoda (2000). "On building prediction systems for software engineers". En: *Empirical Software Engineering* 5.3, págs. 175-182.

Shepperd, Martin y Darrel C Ince (1994). "A critique of three metrics". En: *Journal of Systems and Software* 26.3, págs. 197-210.

Shepperd, Martin y Steve MacDonell (2012). "Evaluating prediction systems in software project estimation". En: *Information and Software Technology* 54.8, págs. 820-827.

Siegel, Jeremy J (2008). *Stocks for the long run*. 4ta ed.

StackExchange (2011). *What does artifact mean?* `https://softwareengineering.stackexchange.com/questions/106473/what-does-artifact-mean`. Accedido el 5/6/2018.

Staron, Miroslaw y col. (2013). "Measuring and visualizing code stability–a case study at three companies". En: *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*. IEEE, págs. 191-200.

Stepanek, G. (2005). *Software project secrets*. Apress.

Stutzke, Richard (2005). *Estimating software-intensive systems: projects, products, and processes*. Addison-Wesley Professional.

Suelmann, Han (2014). "Putnam's Effort-Duration Trade-Off Law: Is the Software Estimation Problem Really Solved?" En: *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*. IEEE, págs. 79-84.

Trendowicz, Adam y Ross Jeffery (2014). "Classification of Effort Estimation Methods". En: *Software Project Effort Estimation*. Springer, págs. 155-208.

UNMSM (2016). *Prospecto de Admisión 2016*. `http://matematicas.unmsm.edu.pe/posgrado/admision.php`. Accedido el 14/1/2019.

Vera, Tomás, Sergio F Ochoa y Daniel Perovich (2017). *Survey of Software Development Effort Estimation Taxonomies*. Inf. téc. Technical Report. Pending ID. Computer Science Department, University of Chile.

Verhoef, Chris (2002). "Quantitative IT portfolio management". En: *Science of computer programming* 45.1, págs. 1-96.

Wang, Yingxu (2003). "On cognitive informatics". En: *Brain and Mind* 4.2, págs. 151-167.

Warburton, Roger D. H. (1983). "Managing and predicting the costs of real-time software". En: *IEEE Transactions on Software Engineering* 5, págs. 562-569.

Wen, Jianfeng y col. (2012). "Systematic literature review of machine learning based software development effort estimation models". En: *Information and Software Technology* 54.1, págs. 41-59.

Whigham, Peter A, Caitlin A Owen y Stephen G Macdonell (2015). "A baseline model for software effort estimation". En: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24.3, pág. 20.

Woodside, C Murray (1979). "A mathematical model for the evolution of software". En: *Journal of Systems and Software* 1, págs. 337-345.

Zambrano, Mauricio (2019). *Normalized Root Mean Square Error*. `https://www.rdocumentation.org/packages/hydroGOF/versions/0.3-10/topics/nrmse`. Accedido el 9/2/2019.

Zheng, Yinhuan y col. (2009). "Estimation of software projects effort based on function point". En: *2009 4th International Conference on Computer Science & Education*. IEEE, págs. 941-943.

Zhou, Yuming y col. (2014). “Source code size estimation approaches for object-oriented systems from UML class diagrams: A comparative study”. En: *Information and Software Technology* 56.2, págs. 220-237.

# ANEXOS

# Medidas de Capacidad Predictiva

Estas medidas se emplean tradicionalmente para comparar la capacidad predictiva de un modelo en el contexto de la estimación de esfuerzo de proyectos de desarrollo de software. Las principales fuentes de este anexo son Shepperd, Cartwright y Kadoda (2000) y Sharma y col. (2010).

## Error Total

Es la suma de diferencias entre los valores que predice el modelo y los valores reales.

$$\sum_{i=1}^{n}(\widehat{x_i} - x_i)$$

Nota: El "residual" es la diferencia entre el valor real y la predicción: $x_i - \hat{x_i}$. Es muy utilizado en el contexto de la regresión lineal.

## Error Absoluto Total

La suma de valores absolutos de los errores.

$$\sum_{i=1}^{n}|\widehat{x_i} - x_i|$$

## MAE (Mean absolute Error) o MAR (Mean absolute resuidual)

El MAE es el promedio de los errores en valor absoluto:

$$\frac{\sum_{i=1}^{n}|\widehat{x_i} - x_i|}{n}$$

El MAR es exactamente el mismo valor, pero desde el punto de vista de los residuales:

$$\frac{\sum_{i=1}^{n}|x_i - \widehat{x_i}|}{n}$$

## ME (Mean Error) o MBE (Mean Bias Error) o Bias

Error promedio de tendencia.

$$\frac{\sum_{i=1}^{n}(\widehat{x_i} - x_i)}{n}$$

Multiplicado por 100 % se obtiene el MPE (Mean percentage error.)

## PRR Predictive-ratio risk

$$\sum_{i=1}^{n} \frac{\widehat{x_i} - x_i}{\widehat{x_i}}$$

## MRE (Magnitude of relative error)

$$|\frac{\widehat{x_i} - x_i}{x_i}|$$

## MMRE (Mean magnitude of relative error)

Media del error total relativo

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} |\frac{\widehat{x_i} - x_i}{x_i}|$$

Es uno de los más utilizados; sin embargo, presenta asimetría en tanto que para un mismo error, los cocientes difieren según si el estimado está por encima o por debajo del valor real.

## MMRE "balanceado" o BRE (balanced relative error)

$$\frac{1}{n}\sum_{i=1}^{n} \frac{|\widehat{x_i} - x_i|}{min(x_i, \widehat{x_i})}$$

## MAPE (Mean absolute percentage error)

$$\frac{100\,\%}{n}\sum_{i=1}^{n} |\frac{\widehat{x_i} - x_i}{x_i}|$$

## PRED(25)

Porcentaje de valores predichos que están dentro del 25 % del valor real. Naturalmente se puede considerar PRED(10), etc. Si N(C) es el número de elementos en el conjunto C entonces:

$$PRED(25) = \frac{100\,\% * N(C)}{n},\ C = \{\widehat{x_i}/\widehat{x_i} \in [x_i - 25\,\%, x_i + 25\,\%]\}$$

## SSE (Sum of Squared Errors)

El "error cuadrático"; permite la comparación de modelos empleando la mismas muestras de un modo muy sencillo. Es la base de los modelos que están a continuación.

$$\sum_{i=1}^{n}(\hat{x_i} - x_i)^2$$

## MSE (Mean Squared Error)

El Error Cuadrático Medio. En términos generales, permite comparar modelos usando respectivamente muestras de distinto tamaño pero con características similares (de preferencia de la misma población.)

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2$$

## RMSE (Root Mean Squared Error)

La raíz cuadrada del anterior. Es similar al anterior, pero tiene la ventaja de estar dado en las mismas dimensiones que la población de estudio.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}$$

## NRMSE (Normalized Root Mean Squared Error)

Es una versión normalizada que se puede emplear para comparar la bondad del ajuste de una función a un conjunto de muestras. Existen diversas fórmulas con el mismo propósito (Zambrano, 2019):

$$\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}}{StdDev(X)}$$

$$\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}}{\overline{X}}$$

$$\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}}{\text{máx}(X) - \text{mín}(X)}$$

donde $StdDev(X)$ es la desviación estándar de los valores de la muestra (conjunto $X = \{x_1 ... x_n\}$); $\overline{X}$ es el promedio de los valores de la muestra; máx$(X)$ y mín$(X)$ corresponden a los valores máximo y mínimo de la muestra.

## TS (Theil statistic)

$$100\,\% * \sqrt{\frac{\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}{\sum_{i=1}^{n} {x_i}^2}}$$

## RE* (Relative Absolute Error)

Relación entre la varianza de los residuales y la varianza de los valores de la muestra.

$$\frac{E[(x_i - \widehat{x_i})^2]}{E[(x_i - \overline{x})^2]}$$

Este coeficiente es positivo. Si el valor promedio $\overline{x}$ se toma como "predicción" para todos los $x_i$ entonces el coeficiente es uno. Cualquier predicción que mejora esta última (en el sentido de obtener un menor error cuadrático) produce un coeficiente menor que uno. Una predicción de mala calidad podría arrojar un valor superior.

## SA (Standard Accuracy)

Refleja qué tan bueno es el modelo en relación a tomar valores al azar.

$$1 - \frac{MAR_M}{\overline{MAR_{Rnd}}}$$

En esta fórmula, $MAR_M$ es el "Mean absolute residual" (ver más arriba) de los valores que el modelo predice. A continuación se obtiene repetidamente (típicamente 1000 veces según Shepperd y MacDonell (2012)) valores para $MAR_{Rnd}$ los cuales son promediados. Los valores $MAR_{Rnd}$ se calculan asignando para $\widehat{x_i}$ a cualquier valor $x_j$ tomado aleatoriamente de la muestra (con $i \neq j$), es decir:

$$(MAR_{Rnd})_r = \frac{\sum_{i=1}^{n} |x_i - x_{rnd(1..n)\neq i}|}{n}$$

Finalmente:

$$\overline{MAR_{Rnd}} = \frac{1}{1000} \sum_{r=1}^{1000} (MAR_{Rnd})_r$$

# Listado de Programas

En esta sección se lista una selección de programas (o extractos de los mismos) a los que se ha hecho referencia en este estudio. La gran mayoría de éstos corresponde a scripts y funciones en lenguaje Octave (se utilizó la versión 4.2.2.)

## Extracción de Datos

El código fuente se ubicó bajo un directorio en sendas subcarpetas (en un sistema Linux.) La generación de las tres primeras columnas de la tabla 3.3 se obtuvo mediante un script de shell bash, aprovechando las herramientas de procesamiento de archivos de texto disponibles en estos sistemas:

```
$ for i in */src ; do
    echo $i ;
    ( wc $(find $i -name '*.java' ) | tail -1 ) ;
done
```

La cuarta columna se genera mediante el script de shell bash:

```
$ for i in */src ; do
    echo $i ;
    ( find $i -name '*.java' | wc ) ;
done
```

### Script de Shell extractor.sh

A continuación el script de shell empleado para la extracción del código fuente (con algunos nombres modificados por cuestiones de confidencialidad):

```
#!/bin/bash
if [ $# != 1 ]
    then
        echo "Proporcionar ruta de proyecto"
        exit
    fi
NAME=proyectoX
SVNURL=http://host/svn/repo/EMPRESA/AREA/trunk/$1
ARCHIVOS=$(svn ls -R $SVNURL/ | grep 'java$')
for A in $ARCHIVOS
```

```
do
    echo "Procesando $A"
    BN=$(basename $A)
    svn log --diff  $SVNURL/$A > /tmp/temporal
    WC_STATS=$(svn cat $SVNURL/$A | wc)
    php extractor-aux.php /tmp/temporal $BN $WC_STATS >>
        resultado-$NAME.csv
done
rm -f /tmp/temporal
```

El comando `svn` cuenta con opciones tales como "`ls -R`" que permite un listado "recursivo" de los archivos de un directorio, así como "`log -diff`" que proporciona todas las versiones del código fuente en un formato que señala las mutuas diferencias entre las versiones. Esto último es muy importante, pues nos permite conocer con precisión el instante en el cual cada archivo de código fuente ha sido alterado (mediante adición, eliminación o cambio.)

Como se aprecia, este script de shell obtiene estadísticas de cada archivo con "`wc`" (variable `WC_STATS`) e invoca a un programa denominado extractor-aux.php que pasamos a listar (con ligeras modificaciones para mejorar la presentación):

## Script PHP extractor-aux.php

```
<?php
if(count($argv) != 6) {
    print "Proporcionar un archivo, alias y wc-stats\n";
    exit;
}
$aname = $argv[1]; $jname = $argv[2]; $wc1 = $argv[3];
$wc2 = $argv[4]; $wc3 = $argv[5];
if(!file_exists($aname)) {
  print "No se encontro archivo ".$aname."\n";
  exit;
}
$fp = fopen($aname, "r");
if(!$fp) {
  print "No se pudo abrir ".$aname."\n";
  exit;
}
$modo = 'I'; // inicio de archivo
$nplus = 0; $nminus = 0;
while($l = fgets($fp)) {
  switch($modo) {
    case 'I':
      if($l != "----------------------------------------\n") {
        print "Data invalida al inicio de ".$argv[1]."\n";
        exit;
      }
      $modo = 'H';
```

```
      break;
    case 'H':
      $parts = explode("|", $l); $version = trim($parts[0]);
      $commiter = trim($parts[1]); $timestamp = trim($parts[2]);
      $tparts = explode("(", $timestamp);
      $stimestamp = trim($tparts[0]);
      $timestamp = strtotime($stimestamp);
      $modo = 'T'; $nplus = 0; $nminus = 0;
    case 'T':
      if($l == "========================================\n") {
        $modo = 'C';
      }
      if($l == "----------------------------------------\n") {
        $modo = 'H';
      }
    break;
    case 'C':
      if($l == "----------------------------------------\n") {
        $modo = 'H';
        print $jname.'|'.$timestamp.'|'.$stimestamp.'|'.
          $version.'|'.$commiter.'|'.$nplus.'|'.
          $nminus.'|'.$wc1.'|'.$wc2.'|'.$wc3."\n";
        break;
      }
      if($l[0] == '+' && $l[1] != "+") {
        $nplus = $nplus + 1;
      }
      if($l[0] == '-' && $l[1] != "-") {
        $nminus = $nminus + 1;
      }
      break;
   }
}
```

El programa procesa el archivo interpretando cuidadosamente el contexto en el que se hace la lectura a partir de ciertos delimitadores que son característicos de "`svn log -diff`". Asimismo contabiliza las líneas agregadas o eliminadas (identificadas por el prefijo "+" y "-".)

## Generación de Datos a Escala diaria

La función Octave `consolida()` extrae los datos de "commits" puntuales y los consolida a escala diaria tomando en consideración los límites de un día físico. Asimismo implementa la eliminación de los datos correspondientes a fines de semana y de los intervalos "en blanco" de larga duración. El resultado se almacena en archivos de nombre "??-diario.csv" cuyo prefijo es el código del proyecto asociado.

```
% consolida data de largo plazo en
```

```
% avances diarios; elimina fines de semana
% asi como espacios sin ninguna actividad de mas
% de cuatro dias (es decir, hace un desplazamiento
% en el tiempo.)
% retorna una matriz con las columnas:
% secuencia-diaria (1,2...), epoch, LOC add, LOC del,
% LOC add-LOC del, LOC add+LOC del
function byday=consolida(project)
dur=duracion(project);
totdias=dur(1);
toskip=dur(2);
fulldata=csvread([project,"-full.csv"]);
dates=fulldata(:,1);
l_added=fulldata(:,2);
l_deleted=fulldata(:,3);
udates=unique(dates);

% fecha inicial (en segundos tras epoch)
ss=localtime(min(dates));
ss.sec=0;
ss.min=0;
ss.hour=0;
min0=mktime(ss);

% saltar contenido de los dias de prototipo
if toskip > 0
    min0=min0+toskip*24*3600;
end
% fecha final, se agrega un dia para truncar por hora cero y
% no perder la ultima informacion
ss=localtime(max(dates)+3600*24);
ss.sec=0;
ss.min=0;
ss.hour=0;
max0=mktime(ss);

% tiempo transcurrido en segundos separados dia a dia
dias=min0:(3600*24):max0;
dias=dias';
byday=[];
mavgw=4;
datainterv=(max0-min0)/(3600*24);
ndiasloop=min([datainterv,totdias]);

% eliminar contenido para los S/D
for d=1:ndiasloop
    diai=dias(d);
    cdiai=localtime(diai);
    wdiai=cdiai.wday;
    diaf=diai+3600*24;
    match=(dates>=diai) & (dates < diaf);
```

```
    if (wdiai == 5 || wdiai == 6)
        l_added(match)=0;
        l_deleted(match)=0;
    end
end
work_day = 0;
for d=1:ndiasloop
    diai=dias(d);
    cdiai=localtime(diai);
    wdiai=cdiai.wday;
    diaf=diai+3600*24;
    emavgw=mavgw;
    % si es jueves o viernes, agregar dos dias al rango de test
    if wdiai+mavgw > 4
        if wdiai+mavgw == 5
            emavgw=mavgw+1;
        else
            emavgw=mavgw+2;
        end
    end
    diaf3=diai+3600*24*emavgw;
    match=(dates>=diai) & (dates < diaf);
    match3=(dates>=diai) & (dates < diaf3);
    tadd3=sum(l_added(match3));
    tdel3=sum(l_deleted(match3));
    if(tadd3+tdel3 < 5)
        continue
    end
    if (wdiai == 5 || wdiai == 6)
        continue
    end
    work_day = work_day + 1;
    tadd=sum(l_added(match));
    tdel=sum(l_deleted(match));
    byday(work_day,1)=work_day;
    byday(work_day,2)=diai;
    byday(work_day,3)=tadd;
    byday(work_day,4)=tdel;
end
% avance neto
byday(:,5)=byday(:,3)-byday(:,4);
% code chunk
byday(:,6)=byday(:,3)+byday(:,4);
end
```

## Gráficos de Frecuencia de la codificación

El siguiente listado implementa el la función `grafico_frecuencia()` en lenguaje Octave que tienen como propósito generar los gráficos en el dominio de la

frecuencia tales como 3.4.11 y 3.4.12:

```
% generacion de grafico de frecuencia de
% actividad de proyecto, opcionalmente eliminando
% fines de semana
function grafico_frecuencia(project,quitar_sd)
dur=duracion(project);
totdias=dur(1);
fulldata=csvread([project,"-full.csv"]);
dates=fulldata(:,1);
l_added=fulldata(:,2);
l_deleted=fulldata(:,3);
udates=unique(dates);

% fecha inicial (en segundos tras epoch)
ss=localtime(min(dates));
ss.sec=0;
ss.min=0;
ss.hour=0;
min0=mktime(ss);

% fecha final, se agrega un dia para truncar por hora cero y
% no perder la ultima informacion
ss=localtime(max(dates)+3600*24);
ss.sec=0;
ss.min=0;
ss.hour=0;
max0=mktime(ss);

% tiempo transcurrido en segundos separados dia a dia
dias=min0:(3600*24):max0;
dias=dias';

byday=[];
work_day = 0;
datainterv=(max0-min0)/(3600*24);
ndiasloop=min([datainterv,totdias]);
for d=1:ndiasloop
    diai=dias(d);
    cdiai=localtime(diai);
    wdiai=cdiai.wday;
    if quitar_sd && (wdiai == 5 || wdiai == 6)
        continue
    end
    diaf=diai+3600*24;
    diaf3=diai+3600*24;
    match=(dates>=diai) & (dates < diaf);
    work_day = work_day + 1;
    tadd=sum(l_added(match));
    tdel=sum(l_deleted(match));
    byday(work_day,1)=work_day;
    byday(work_day,2)=diai;
```

```
    byday(work_day,3)=tadd;
    byday(work_day,4)=tdel;
end
x=byday(:,3);
y=fft(x);
l=length(y);
f=((0:l-1)/l)';
range=floor(0.05*l:0.5*l);
f=f(range,1);
y=y(range,1);
hf=figure();
plot(1./f,abs(y));
ylim([0 40000]);
xlabel("Tiempo en dias");
ylabel("Productividad LOC/dia");
if quitar_sd
    title('Grafico de frecuencia sin S/D');
    saveas(hf,"../DOC/freq_sin_sd.png");
else
    title('Grafico de frecuencia L-D');
    saveas(hf,"../DOC/freq_ld.png");
end
end
```

# Aproximación Runge-Kutta

El siguiente listado implementa el método de Runge-Kutta de cuarto orden para ecuaciones diferenciales en lenguaje Octave:

```
function [y,t] = rk4(f,a,b,y0,n)
h = (b-a)/n;
h2 = h/2;
y(1,:) = y0;
t(1,:) = a;
for i = 1 : n
    t(i+1,:) = t(i) + h;
    y2 = t(i) + h2;
    s1 = f(y(i,:), t(i));
    s2 = f(y(i,:) + h2 * s1, y2);
    s3 = f(y(i,:) + h2 * s2, y2);
    s4 = f(y(i,:) + h * s3, t(i+1));
    y(i+1,:) = y(i,:) + (s1 + 2*s2 + 2*s3  + s4) * h/6;
end
```

**Definición de EDO** En Octave se implementa la función productividad() correspondiente al segundo miembro de la ecuación 4.2.2:

```
function r=productividad(a,k,D0,N0,N)
    d=1-(1-D0)*exp(-a*(N-N0));
```

```
    r=k/d;
end
```

La aproximación Runge-Kutta se debe invocar con esta función mediante argumento funcional:

```
octave> [y,t]=rk4(@(py,pt) productividad(a,k,D0,N0,py),0,Tf,N0,200);
```

# Evaluación de modelo ELI

## Función de Error

Función de error $e_i(a, k, D_0, N_0)$ de una muestra con ELI:

```
function r=termdelta(a,k,D0,N0,ti,Ni)
    Nt=from_t_n(a,k,D0,N0,ti);
    r=(Nt-Ni);
endfunction
```

Función de error total mediante suma de términos de error individual $E(a, k, D_0, N_0)$:

```
function r=rdelta(a,k,D0,N0,T,N)
    r=0.0;
    for i=1:length(T)
        td=termdelta(a,k,D0,N0,T(i),N(i));
        r=r+td*td;
    end
    r=r/10000;
endfunction
```

## Minimización y Predicción de Datos

Minimización de ELI a partir de datos en columnas: 1=tiempo; 2=neto (agregados menos eliminados); 3=chunk (agregados más eliminados.) Observar que la función de minimización no emplea directamente la función de error total `rdelta()`, sino una versión escalada en el primer argumento:

$$rdelta^*(a, b, c, d) = rdelta(\frac{a}{10^6}, b, c, d)$$

Esto se hizo puesto que el empleo directo de `rdelta()` generó resultados insatisfactorios (lo que fue detectado al examinar las gráficas resultantes.) El primer argumento es un valor que tiende a ser muy reducido (en la escala de $10^{-6}$), lo cual suele generar inconvenientes en la aproximación numérica de diferenciales (en nuestro caso, para el gradiente de la optimización); esto nos condujo al uso de la función alternativa que solucionó el inconveniente.

```
% encuentra datos de funcion eli ajustada; proporcionar
% datos semanales y la columna (2 o 3) de interes
% correspondiente a ADD-DEL o ADD+DEL
function r=eli_data(cdata,col)
t=cdata(:,1)*7;
N=cdata(:,col);

Tf=(length(t)-1)*7;
Nf=N(end);
N0=N(1);

% aproximacion para los parametros: primer punto de minimizacion
k_d0=(N(2)-N(1))/7;
if k_d0 <= 0
    k_d0=1;
    fprintf('Forzando␣kd0=%g\n',k_d0);
end
alpha=4*(k_d0*Tf/Nf-1);
if alpha <= 0
    alpha=1;
end
a=2/Nf;
D0=a*exp(-a*N0)*Nf/alpha;
k=k_d0*D0;

fprintf("Punto inicial: a=%g k=%g D0=%g N0=%g\n",a,k,D0,N0);
fprintf("Minimo inicial: %g\n",rdelta(a,k,D0,N0,t,N));

% optimizar buscando minimos cuadrados
factor=1000000;

% optimizar buscando minimos cuadrados
[x, obj, info, iter, nf, lambda] = sqp([a*factor;k;D0;N0],
    @(v) rdelta(v(1)/factor,v(2),v(3),v(4),t,N,500,1e-9),
    [],[],
    [1e-10;1e-9;1e-9;0.1*N0],
    [100*a*factor;100*k;100*D0;10*N0]);
x(1)=x(1)/factor;
fprintf("Punto final: %g %g %g %g\n",x(1),x(2),x(3),x(4));
fprintf("Minimo final: %g\n",rdelta(x(1),x(2),x(3),x(4),t,N));

% obtener los valores mediante el modelo (prediccion)
tx=[];
nx=[];
for i=1:length(t)
    tx(i)=t(i);
    nx(i)=from_t_n(x(1),x(2),x(3),x(4),t(i));
end
r=nx';
endfunction
```

**ELI en forma implícita** Corresponde a un despeje de "$t$" a partir de 4.2.3, que anteriormente fue denominado $t = T(N)$.

```
function r=from_n_t(a,k,D0,N0,Ni)
    kti=(Ni-N0)+(1-D0)*(exp(-a*(Ni-N0))-1)/a;
    %kti=(D0+exp(-a*N0))*(Ni-N0)-(1.0/a)*(exp(-a*N0)-exp(-a*Ni));
    r=kti/k;
endfunction
```

**ELI en forma explícita** Obtención de $N(t)$; se implementa mediante la solución de la ecuación auxiliar

$$T(N) - t = 0$$

esta función se emplea para obtener el valor "predicho" por el modelo tras haberse hallado sus parámetros:

```
function r=from_t_n(a,k,D0,N0,t)
    if t == 0
        r=N0;
    else
        if (from_n_t(a,k,D0,N0,0.95*N0)-t)* (from_n_t(a,k,D0,N0,1000000)-t) >= 0
            r=10000;
        else
            r=fzero(
                @(x)(from_n_t(a,k,D0,N0,x)-t),
                [0.95*N0,1000000]);
        end
    end
endfunction
```

## Obtención de NMRSE y MAE

El siguiente script permite obtener estas métricas a partir de los valores modelados almacenados en sendos archivos:

```
% Generacion de NRSME y MAE
projects={'sp';'re';'ad';'as';'jp';'ct';'po'};
nom_meth={'ELI';'CUAD';'CUB';'ELD';'LLD'};
MSE=[];
NRMSE1=[];
MAE=[];
NP=[];
DERI=[]; DERF=[];
SIZEP=[];
for i=1:length(projects)
    project=projects{i,1};
    filename=["../OCTAVE-DATA/", project, "-semanal.csv"];
    cdata=csvread(filename);
    col_data=5;
```

```
    ncdata=[cdata(:,1),cumsum(cdata(:,col_data))];
    muestra=ncdata(:,2);
    NP(i,1)=length(muestra);
    STDM=std(muestra);
    PROM=mean(muestra);
    MAXMIN=max(muestra)-min(muestra);
    SIZEP(i,1)=muestra(end);
    for nm=1:length(nom_meth)
        mname=nom_meth{nm,1};
        f_in=[project,"-",mname,"-model.dat"];
        load(f_in);
        eval(["model=model_",project,"_",mname,";"]);
        dif2=(muestra-model).^2;
        difabs=abs(muestra-model);
        MSE(i,nm)=sum(dif2);
        NRMSE1(i,nm)=100*sqrt(sum(dif2)/NP(i,1))./STDM;
        % alternativas a NRMSE
        % NRMSE2(i,nm)=100*sqrt(sum(dif2)/NP(i,1))./PROM;
        % NRMSE3(i,nm)=100*sqrt(sum(dif2)/NP(i,1))./MAXMIN;
        MAE(i,nm)=mean(difabs);
        dmodel=diff(model);
        DERI(i,nm)=dmodel(1);
        DERF(i,nm)=dmodel(end);
    end
end
elif=DERF(:,1);
elii=DERI(:,1);
csvwrite('lambda.csv',[elii,elif,SIZEP]);
DeltaProd=100*(elii-elif)./elii;
DeltaProdEsp=DeltaProd./log(SIZEP);
NRMSE=[NRMSE1,NP]
MAE=[MAE,NP]
DERI
DERF
SIZEP
DeltaProd
DeltaProdEsp

% ajuste de modelo lineal YSZ=SIZEP o log(SIZEP)
YSZ1=sqrt(SIZEP);

% ajuste
n=length(elii);
X=[ones(n,1),elii];
[BETA, SIGMA, R] = ols(YSZ1,X);
y_average=mean(YSZ1);
ny2 = n*y_average*y_average;
ypy = sum(YSZ1.^2);
betapxpy = (BETA')*(X')*YSZ1;
rsq = (betapxpy - ny2)/(ypy - ny2);
printf('R^2=%g␣SSE=%g\n',rsq, sum(R.^2));
```

```
save('size-lin-model.dat',"BETA","SIGMA","R");
```

## Test Wilcoxon

La siguiente función Octave se utilizó para implementar el test Wilcoxon de rango con signo para exactamente siete muestras (pues disponemos de siete proyectos.) Para un número mayor de muestras, se debe implementar la consulta a la tabla de la estadística T de Corder y Foreman (2014, p.244); para 25 o más muestras, simplemente invocar a la función incorporada `wilcoxon_test()`.

```
function little_wilcoxon (x, y)
if (! (isvector (x) && isvector (y) && (length (x) == length (y))))
    error ("wilcoxon_test: X and Y must be vectors of the same length");
endif
n = length (x);
x = reshape (x, 1, n);
y = reshape (y, 1, n);
d = x - y;
d = d(find (d != 0));
n = length (d);
r = ranks (abs (d));
z = sum (r(find (d > 0)));
z2 = sum (r(find (d < 0)));
T=min(z,z2);
fprintf('T=%g\n',T);
if n != 7
    error('Este␣test␣solo␣opera␣con␣7␣muestras');
end
if T < 2
    fprintf('Se␣rechaza␣hipotesis␣nula\n',T,n);
end
if T == 2
    fprintf('T␣en␣valor␣critico␣-␣no␣concluyente\n',T,n);
end
if T > 2
    fprintf('No␣se␣rechaza␣hipotesis␣nula\n',T,n);
end
endfunction
```

## Predicción vía tendencia ideal

La función Octave mostrada a continuación se utilizó para la generación de las predicciones de las tablas 4.8 y 4.9. Implementa el algoritmo descrito en la sección 4.2.6.2.

```
% prediccion de duracion via tendencia ideal
% emplea archivos de data semanal
function pred(project, method, nsem, target_size)
```

```
if exist("target_size","var") != 1
    target_size=-1;
end
printf('Target␣=%d\n',target_size);
switch(method)
    case 'ELI'
        funmodelo=@from_t_n;
    case 'CUAD'
        funmodelo=@polyval;
    case 'CUB'
        funmodelo=@polyval;
    case 'ELD'
        funmodelo=@feld;
    case 'LLD'
        funmodelo=@flld;
    otherwise
        printf('Modelo␣erroneo\n');
        return
endswitch
load('size-lin-model.dat');
sbeta1=BETA(1);
sbeta2=BETA(2);
load('prod-lin-model.dat');
beta1=BETA(1);
beta2=BETA(2);
beta3=BETA(3);
filename=["../OCTAVE-DATA/", project, "-semanal.csv"];
cdata=csvread(filename);
col_data=5;
ncdata=[cdata(:,1),cumsum(cdata(:,col_data))];
params=csvread(["partial_",project,"_",method,"_",num2str(nsem),".csv"]);
last=0;
der0=0;
TOP_SEMANAS=500;
vajuste=[];
for i=1:TOP_SEMANAS
    fparams=params;
    fparams(end+1)=i*7;
    if strcmp(method,"CUAD") || strcmp(method,"CUB")
        ajuste=funmodelo(params,i*7);
    else
        ajuste=funmodelo(num2cell(fparams){:});
    end
    vajuste(i,1)=ajuste;
    der=ajuste-last;
    if i==1
        der0=der;
        if target_size == -1
            target_size=sbeta1+sbeta2*der0;
            printf("Target size ajustado de modelo lineal=%g\n",target_size);
        end
```

```
    else
        if target_size != -1
            if ajuste >= target_size
                printf("> Prediccion %s muestra=%d: %d semanas.\n",project,nsem,i);
                printf('Semana␣%d␣size-pred=%g␣der-ELI=%g␣target=%g\n',
                    i,ajuste,der,target_size);
                hf=figure();
                times_mat=cdata(:,1);
                %times_mat=cdata(:,1)*7;
                col_data=5;
                ncdata=[cdata(:,1),cumsum(cdata(:,col_data))];
                data=ncdata(:,2);
                load([project,"-",method,"-model.dat"]);
                eval(["model=model_",project,"_",method,";"]);
                plot(times_mat,data,'*;Sample;',times_mat,model,'-;Fit;',1:rows(vajust
                xlabel('Semanas');
                ylabel('N(t)');
                title(['Prediccion␣Ideal␣',method, ':␣', project,'␣desde␣',num2str(nse
                saveas(hf,['../DOC/pred-ideal-',project,'-',method,'-',num2str(nsem),
                return
            end
        else
            error('Error␣interno.');
        end
    end
    last=ajuste;
end
printf('>␣No␣se␣encontro␣prediccion␣para␣%s␣muestra=%d\n',project,nsem);
end
```