TESIS DE MAESTRÍA

Ingeniería en Sistemas de Información

“Estudio del comportamiento dinámico de sistemas de información basado en redes complejas”

Autor: Jorge Benjamín Pagani

Director de Tesis: Dr. Hernán Merlino

Co director de Tesis: Dr. Darío Rodriguez

Buenos Aires - 2015

DEDICATORIA

A Jennifer, por hacerme feliz;

A mis padres, Roberto y Milagros, quienes con su sacrificio me permitieron llegar a donde estoy;

A mis hermanas, Socorro y Milagros, por acompañarme siempre;

A mis abuelos, Jorge e Inés, por todo el apoyo que me dan.

RESUMEN

La Ingeniería del Software requiere de métricas adecuadas para dar soporte a la toma de decisiones técnicas y de gestión en el proceso de desarrollo de software. Se han desarrollado cientos de métricas hasta el momento pero el área de métricas de software todavía no se encuentra en un nivel de madurez adecuado para una disciplina sistémica y cuantificable como la Ingeniería del Software. Esto se debe principalmente a que la mayoría de las métricas desarrolladas no proveen la información necesaria de forma oportuna y precisa. En esta tesis de Maestría el autor propone una metodología de análisis dinámico del diseño de sistemas basada en redes complejas, consistente en el modelado del diseño de sistemas como una red compleja y en el cálculo de métricas sobre dicho modelo.

ABSTRACT

Software engineering requires adequate metrics to support the technical and management decision-making processes in the development of software. Hundreds of metrics have been developed, but the software metrics area has not yet reached the level of maturity required in a systemic and measurable discipline such as software engineering. This is mainly because most of the developed metrics do not provide necessary information in an accurate and timely manner. In this master's thesis, the author proposes a methodology for the dynamic analysis of a system's design based on complex networks. The dynamic analysis consists of modeling the system's design as a complex network and the calculation of metrics on that model.

AGRADECIMIENTOS

A la Escuela de Posgrado de la Universidad Tecnológica Nacional - Facultad Regional Buenos Aires por haberme abierto sus puertas para realizar mis estudios de Maestría.

A las Facultades Regionales de Resistencia y de Buenos Aires de la Universidad Tecnológica Nacional por brindar mi formación profesional de grado.

Al profesor Dr. Ramón García-Martínez, cuya desinteresada ayuda fue transcendental para el desarrollo de esta tesis.

A mis directores de tesis, Dr. Hernán Merlino y Dr. Darío Rodriguez, quienes me guiaron a lo largo del desarrollo de este trabajo.

Al Laboratorio de Investigación y Desarrollo en Arquitecturas Complejas, perteneciente al Grupo de Investigación de Sistemas de Información de la Universidad Nacional de Lanús, por permitirme desarrollar las investigaciones vinculadas a esta tesis.

A mis padres, Roberto y Milagros, por haberme dado el soporte financiero para realizar mis estudios de maestría.

A mi madre, la Lic. Milagros Delfino, quien realizó decenas de sugerencias lingüísticas para facilitar la lectura y comprensión de esta obra.

ÍNDICE

1. INTRODUCCIÓN 1
1.1. Contexto de la tesis 1
1.2. Objetivos de la tesis 2
1.2.1. Objetivo general 2
1.2.2. Objetivos específicos 2
1.2.3. Alcance de los objetivos 3
1.3. Metodología de desarrollo de la tesis 3
1.4. Producciones científicas vinculadas a esta tesis 4
1.5. Estructura general de la tesis 4
2. ESTADO DE LA CUESTIÓN 7
2.1. Proceso unificado de desarrollo de software 7
2.2. Diagramas UML para el flujo de análisis y diseño de RUP 10
2.3. Métricas de software 14
2.4. Métricas dinámicas 20
2.4.1. Definición de métricas dinámicas y diferencias con métricas estáticas 20
2.4.2. Métricas dinámicas de acoplamiento 22
2.4.2.1. Métricas EOC e IOC 22
2.4.2.2. Métricas de Arisholm 23
2.4.2.3. Métricas de Mitchell y Power 24
2.4.2.4. Métrica dinámica de acoplamiento (DCM) 25
2.4.2.5. Comparativa de métricas dinámicas de acoplamiento 25
2.4.3. Métricas dinámicas de cohesión 26
2.4.3.1. Métricas de Gupta y Rao 26
2.4.3.2. Métricas de Mitchell y Power 26
2.4.3.3. Métricas de Gupta y Chhabra 28
2.4.3.4. Comparativa de métricas dinámicas de cohesión 29
2.4.4. Métricas dinámicas de complejidad 29
2.4.4.1. Métricas de Munson y Khoshgoftaar 29
2.4.4.2. Métricas de Yacoub et al. 30
2.4.4.3. Métricas de Mathur et al. 30
2.4.4.4. Comparativa de métricas dinámicas de complejidad 31
2.4.5. Métricas dinámicas de polimorfismo 31
2.4.5.1. Métricas de Dufour et al. 31

2.4.5.2. Métricas de Choi y Tempero 32
2.4.5.3. Métricas de Sandhu y Singh 33
2.4.6. Otras métricas dinámicas 33
2.4.7. Métricas pseudodinámicas 33
2.5. Métricas a nivel de diseño 34
2.5.1. Definición de métricas a nivel de diseño y diferencias con métricas… 34
2.5.2. Métricas a nivel de sistema 37
2.5.2.1. MHF 37
2.5.2.2. AHF 38
2.5.2.3. MIF 38
2.5.2.4. AIF 38
2.5.2.5. PF 39
2.5.2.6. CF 39
2.5.3. Métricas a nivel de acoplamiento y uso 40
2.5.3.1. CBO 40
2.5.4. Métricas a nivel de herencia 40
2.5.4.1. DIT 41
2.5.4.2. NOC 41
2.5.4.3. SIX 41
2.5.4.4. ANA 41
2.5.5. Métricas a nivel de clase 42
2.5.5.1. RFC 42
2.5.5.2. WMC 42
2.5.5.3. LCOM 42
2.5.5.4. DAC 43
2.5.5.5. DAC’ 43
2.5.5.6. NOM 43
2.5.5.7. SIZE2 43
2.5.5.8. APPM 44
2.5.6. Métricas a nivel de métodos 44
2.5.6.1. LOC 44
2.5.6.2. NOM 44
2.5.6.3. CAMC 45
2.6. Métricas para UML 45
2.6.1. Métricas de Kim y Boldyreff 45

2.6.1.1. Métricas para modelos 45
2.6.1.2. Métricas para clases 46
2.6.1.3. Métricas para mensajes 47
2.6.1.4. Métricas para casos de uso 48
2.6.2. Métricas de Marchesi 48
2.6.2.1. Métricas de clase 49
2.6.2.2. Métricas de paquete 49
2.6.2.3. Métricas de sistema 49
2.6.3. Métricas de Genero 50
2.6.3.1. Métricas de diagrama de clase 50
2.6.3.2. Métricas de clase 51
2.7. Redes complejas 51
2.7.1. Introducción a las redes complejas 52
2.7.2. Sistemas software como redes complejas 54
2.7.3. Métricas de software mediante redes complejas 57
3. DESCRIPCIÓN DEL PROBLEMA 59
3.1. Importancia de la medición en el proceso de desarrollo de software 59
3.2. Identificación del problema de investigación 60
3.3. Descripción del problema 61
3.4. Sumario de investigación 61
4. SOLUCION PROPUESTA 63
4.1. MADDS 63
4.1.1. Generalidades 63
4.1.2. Descripción de MAADS 64
4.1.3. Integración de MAADS dentro del proceso unificado de desarrollo 66
4.2. Modelado del comportamiento del sistema mediante redes complejas 67
4.3. Cálculo de métricas orientadas al diseño de sistemas 73
4.3.1. Métricas propuestas en esta tesis 73
4.3.1.1. Cantidad de nodos 73
4.3.1.2. Cantidad de aristas 74
4.3.1.3. Llamadas totales 74
4.3.1.4. Promedio de llamadas por método 74
4.3.1.5. Promedio de llamadas por clase 75
4.3.1.6. Métodos sin uso 75
4.3.1.7. Métodos con un único uso 75

4.3.1.8. Métodos muy utilizados 76
4.3.1.9. Tendencia central de la complejidad 76
4.3.1.10. Complejidad dinámica total del sistema 78
4.3.1.11. Cohesión 78
4.3.1.12. Polimorfismo 79
4.3.1.13. Sobrecarga 80
4.3.1.14. Acoplamiento 80
4.3.1.15. Herencia 81
4.3.1.16. Resumen de métricas propuestas en esta tesis 81
4.3.2. Métricas existentes en otros dominios, reinterpretadas al dominio de sistemas 82
4.3.2.1. Factor small-world 82
4.3.2.2. Hubbing 83
4.3.2.3. Factor scale-free 83
4.3.2.4. Coeficiente de agrupamiento 84
4.3.2.5. Resumen de métricas existentes en otros dominios 85
5. VALIDACIÓN 87
5.1. Introducción al método de validación 87
5.2. Aplicación del método de validación 88
5.3. Estudio de las métricas de MADDS 89
5.3.1. Variables independientes 89
5.3.2. Variables dependientes 90
5.3.3. Validación de métricas desarrolladas en esta tesis 91
5.3.3.1. Métrica del promedio de llamadas por método 91
5.3.3.2. Métrica del promedio de llamadas por clase 92
5.3.3.3. Métrica de métodos muy utilizados 94
5.3.3.4. Métrica de tendencia central de la complejidad 95
5.3.3.5. Métrica de complejidad dinámica total del sistema 97
5.3.3.6. Métrica de cohesión 97
5.3.3.7. Métricas de polimorfismo y sobrecarga 99
5.3.3.8. Métrica de acoplamiento 101
5.3.3.9. Métrica de herencia 102
6. CONCLUSIONES 105
6.1. Aportes de esta tesis 105
6.2. Futuras líneas de investigación 106
7. REFERENCIAS 109

ÍNDICE DE FIGURAS

Figura 2.1	Ciclo de vida del proceso unificado de Rational	8
Figura 2.2	Flujo de trabajo de RUP para el Análisis y Diseño	10
Figura 2.3	Diagramas de comportamiento UML v2.x	11
Figura 2.4	Curva ROC para la detección de errores de software	36
Figura 2.5	Función de distribución potencial del tipo P(k)~k^(-y)	53
Figura 2.6	Visualización de una clase de VTK como red compleja [Myers, 2003]	55
Figura 3.1	Distribución de métricas de diseño por enfoque [Tahir y MacDonell, 2012]	61
Figura 4.1	Flujo de trabajo de RUP con MADDS para el Análisis y Diseño	65
Figura 4.2	Actividades de MADDS	65
Figura 4.3	Relación entre artefactos de MADDS	67
Figura 4.4	Diagrama de secuencia UML	68
Figura 4.5	Resultado del primer paso de MADDS	69
Figura 4.6	Resultado del paso (c) de MADDS	70
Figura 4.7	Resultado final del subproceso de modelado de MADDS	72
Figura 4.8	Casos de uso (a) y (b)	73
Figura 5.1	Comportamiento de la métrica de métodos muy utilizados al modificar E_v	95
Figura 5.2	Comportamiento de la métrica de cohesión al modificar el umbral U_c	99

ÍNDICE DE TABLAS

Tabla 2.1	Principales relaciones entre clases y su modelado en UML v2 [OMG, 2011]	12
Tabla 2.2	Principales nodos de UML v2.4.1 [OMG, 2011]	13
Tabla 2.3	Comparación entre métricas estáticas y métricas dinámicas	21
Tabla 2.4	Métricas dinámicas de acoplamiento de Arisholm	23
Tabla 4.1	Relación entre componentes del diagrama de secuencia y la red compleja	67
Tabla 4.2	Resultado del segundo paso de MADDS	70
Tabla 4.3	Resultado del paso c de MADDS	70
Tabla 4.4	Resultado del paso d de MADDS	71
Tabla 4.5	Resumen de métricas propuestas en esta tesis	81
Tabla 4.6	Resumen de métricas existentes en otros dominios	85
Tabla 5.1	Listado de variables independientes	89
Tabla 5.2	Listado de variables dependientes	90
Tabla 5.3	Variables experimentales de la métrica del promedio de llamadas por método	91
Tabla 5.4	Valores definidos para las variables experimentales de la métrica del promedio de llamadas por método	92
Tabla 5.5	Variables experimentales de la métrica del promedio de llamadas por clase	93
Tabla 5.6	Valores definidos para las variables experimentales de la métrica del promedio de llamadas por clase	93
Tabla 5.7	Variables experimentales de la métrica de métodos muy utilizados	94
Tabla 5.8	Valores definidos para las variables experimentales de la métrica de métodos muy utilizados	94
Tabla 5.9	Variables experimentales de la métrica de tendencia central de la complejidad	96

Tabla 5.10 Valores definidos para las variables experimentales de la métrica de tendencia central de la complejidad 96

Tabla 5.11 Variables experimentales de la métrica de complejidad dinámica total del sistema 97

Tabla 5.12 Valores definidos para las variables experimentales de la métrica del promedio de llamadas por método 97

Tabla 5.13 Variables experimentales de la métrica de cohesión 98

Tabla 5.14 Valores definidos para las variables experimentales de la métrica de cohesión 98

Tabla 5.15 Variables experimentales de las métricas de polimorfismo y sobrecarga 100

Tabla 5.16 Valores definidos para las variables experimentales de las métricas de polimorfismo y sobrecarga 100

Tabla 5.17 Variables experimentales de la métrica de acoplamiento 101

Tabla 5.18 Valores definidos para las variables experimentales de la métrica de acoplamiento 102

Tabla 5.19 Variables experimentales de la métrica de herencia 102

Tabla 5.20 Valores definidos para las variables experimentales de la métrica de herencia 103

NOMENCLATURAS Y ACRÓNIMOS

ANSI	American National Standards Institute
CDN	Component Dependency Network
CIF	Common Industry Format
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ISSN	International Standard Serial Number
MDA	Model Driven Architecture
MDD	Model Driven Development
OMG	Object Management Group
ROC	Receiver Operating Characteristic
ROOM	Real-Time Object-Oriented Modeling
RUP	Rational Unified Process
RUSP	Ready to Use Software Product
UML	Unified Modeling Language

1. INTRODUCCIÓN

En este capítulo se describe el contexto de esta tesis y se definen los problemas abiertos identificados (sección 1.1). A partir de los mismos, se presentan los objetivos definidos para la tesis (sección 1.2) y se describe la metodología utilizada para el desarrollo del trabajo (sección 1.3). Finalmente, se enumeran las producciones científicas vinculadas al desarrollo de la tesis (sección 1.4) y se realiza una breve descripción de la estructura de la misma (sección 1.5).

1.1 Contexto de la tesis

La Ingeniería del Software, al igual que todas las ramas de la Ingeniería, es una disciplina inherentemente cuantitativa. Por ende, requiere de formas de medición para comprender las características de sus procesos y productos con el fin de construir software de calidad.

La medición, implementada mediante métricas, permite a los ingenieros de software evaluar mediante criterios objetivos las diversas características y atributos de los procesos y productos involucrados en el desarrollo de software, brindándoles información de valor para los procesos de toma de decisiones, tanto en lo que respecta a aspectos técnicos como a aspectos de gestión [Chhabra y Gupta, 2010].

En lo que respecta a aspectos de gestión, las métricas de software permiten a los ingenieros realizar el planeamiento y control del proceso de desarrollo mediante métricas de proceso y métricas de proyecto. Los aspectos técnicos del desarrollo de software, en cambio, son evaluados mediante las métricas de producto, las cuales permiten estudiar los atributos, tanto internos como externos, del software [Bellini *et al.*, 2008].

Las métricas de software posibilitan entonces cuantificar aspectos del proceso o del proyecto de desarrollo de software como el tiempo de desarrollo y el costo del mismo, como así también aspectos del producto software como su tamaño y su complejidad. La información que se obtiene de las métricas brinda el apoyo que requieren los ingenieros para construir software de calidad [Pressman, 2005].

Si bien las métricas de software tienen varias décadas de desarrollo, la mayoría de los autores coincide en que las mismas no se encuentran en un nivel de madurez adecuado [Kitchenham, 2010; Bellini *et* al, 2008; Kaner, 2004; Fenton y Neil, 2000; Glass, 1994]. La mayoría de las métricas desarrolladas están orientadas a medir atributos del producto software ya desarrollado en lugar de ofrecer la información necesaria para la toma de decisiones durante el ciclo de vida del desarrollo de software [Fenton y Neil, 2000].

La etapa de diseño del sistema requiere, en particular, de métricas adecuadas para guiar el proceso de diseño [Pressman, 2005]. No solo es importante disponer de métricas para diseñar artefactos óptimos sino que también es de suma importancia minimizar los errores en la etapas tempranas del desarrollo de software, ya que los errores en el desarrollo normalmente son propagados a etapas posteriores en donde son más costosos de identificar y corregir [Boehm, 1981].

Asimismo, es necesario que las métricas reflejen el comportamiento real de los artefactos bajo estudio, aspecto que solo puede medirse mediante las métricas dinámicas [Tahir y MacDonell, 2012]. Las métricas estáticas, de mayor presencia en la literatura, todavía no han demostrado su habilidad para capturar el comportamiento real de una aplicación [Chhabra y Gupta, 2010] y se encuentran limitadas a la hora de evaluar características del paradigma orientado a objetos como el uso de clases y el polimorfismo.

Se ha señalado la necesidad de disponer de un conjunto de métricas dinámicas a nivel de diseño [Tahir y MacDonell, 2012] que permitan obtener información precisa y oportuna para guiar el proceso de desarrollo de software.

1.2 Objetivos de la tesis

En esta sección se presentan los objetivos planteados para el desarrollo de esta tesis. Los objetivos son divididos en un objetivo general que representa la finalidad de este trabajo (sección 1.2.1) y en un conjunto de objetivos específicos derivados del objetivo general (sección 1.2.2). Asimismo se describe el alcance de los objetivos planteados (sección 1.2.3).

1.2.1 Objetivo general

El objetivo general de este trabajo es desarrollar una metodología de análisis que permita estudiar y evaluar el comportamiento dinámico de un sistema software desde el punto de vista del diseño de sistemas.

1.2.2 Objetivos específicos

Se plantean en esta sección una serie de objetivos específicos, los cuales estarán asociados al desarrollo de la metodología mencionada en el objetivo general. Los objetivos específicos son:

- Desarrollar un proceso de modelado del comportamiento de un sistema software basado en redes complejas que tenga como punto de partida diagramas de interacción de UML v2
- Desarrollar un conjunto de métricas, aplicables sobre los modelos generados, que permitan analizar y evaluar el comportamiento del software a partir de los principios del diseño de sistemas

- Analizar y estudiar el comportamiento de las métricas propuestas utilizando un método de validación empírico por simulación

1.2.3 Alcance de los objetivos

En esta sección se describe el alcance de los objetivos presentados previamente:

- El proceso a desarrollar debe brindar una concepción integral del software pero a la vez permitir que sea aplicado a una sola parte del mismo
- La metodología a desarrollar debe ser escalable a fin de permitir la incorporación de nuevas métricas
- Las métricas a desarrollar deben cumplir con los criterios de validación de métricas de software definidos en el estándar IEEE 1061-1998

1.3 Metodología de desarrollo de la tesis

En esta sección se describen los pasos metodológicos correspondientes con el desarrollo de esta tesis, en la cual se siguió un enfoque de investigación clásico [Creswell, 2002; Rosas y Riveros, 1985] con el objetivo de desarrollar una metodología de estudio del comportamiento dinámico de sistemas de información basado en redes complejas.

El primer paso consiste en una investigación documental sobre métricas de software, particularmente en lo que respecta a métricas dinámicas y a métricas a nivel de diseño, y al modelado de sistemas software como redes complejas, realizando revisiones sistémicas de los artículos científicos [Argimón, 2004] en función de los objetivos propuestos en esta tesis.

En el segundo paso, y a partir de la investigación documental, se avanza en el desarrollo de un prototipo evolutivo experimental [Basili, 1993] para dar cumplimiento a los objetivos de la tesis, es decir, para desarrollar una metodología de estudio del comportamiento dinámico de sistemas de información basada en redes complejas. Este prototipo evolutivo experimental será refinado continuamente a lo largo del desarrollo de la tesis mediante su aplicación a casos de estudio de complejidad creciente.

El prototipo elaborado define dos actividades principales las cuales son desarrolladas independientemente de forma tal de cumplir con los objetivos de la tesis. En el siguiente paso se propone la primera actividad, la cual consiste en el modelado de sistemas software como redes complejas y, en un paso posterior, la segunda actividad en la cual se generan un conjunto de métricas aplicables sobre el modelo desarrollado con el objetivo de estudiar el comportamiento sistemas software. El trabajo en ambas actividades es realizado considerando formalismos usuales del

modelado conceptual en la Ingeniería del Software [Hossian, 2012; Rumbaugh *et al.*, 1999] y los modelos de proceso habituales en la Ingeniería de Software [ANSI/IEEE, 2007; Oktaba *et al.*, 2007; IEEE, 1997].

Posteriormente, y mediante la aplicación de un método de Monte Carlo, se realiza el análisis y estudio del comportamiento de las métricas desarrolladas, continuando con el refinamiento del prototipo evolutivo experimental desarrollado previamente hasta alcanzar la estabilización del prototipo. El proceso de refinamiento implica modificaciones sobre ambas actividades de la metodología propuesta.

Por último, se elabora un informe final en el cual se presenta la metodología propuesta y se indican futuras líneas de investigación en el análisis del comportamiento dinámico de sistemas software.

1.4 Producciones científicas vinculadas a esta tesis

Durante el desarrollo de esta tesis se presentaron dos trabajos con los resultados parciales que fueron obtenidos. Uno de los trabajos fue publicado en una revista y el otro es un trabajo de especialización. A continuación se citan dichas producciones:

Pagani, J. B., (2015). *Investigación en progreso: Estudio del Comportamiento Dinámico del Diseño de Sistemas*. Revista Latinoamericana de Ingeniería de Software, Volumen 2, Número 1, Páginas 95-98. ISSN 2314-2642.

Pagani, J. (2016). Estudio del Comportamiento Dinámico de Sistemas de Información Basado en Redes Complejas. Revista Latinoamericana de Ingeniería de Software, Volumen 4, Número 1 (en prensa). ISSN 2314-2642.

1.5 Estructura general de la tesis

La tesis se estructura en siete capítulos: Introducción, Estado de la cuestión, Descripción del problema, Solución propuesta, Validación, Conclusiones y Referencias. A continuación se realiza una breve descripción de los mismos.

En el primer capítulo, *Introducción*, se plantea el contexto de la tesis y se mencionan los problemas abiertos identificados. Además se presentan los objetivos de la tesis y la metodología utilizada para el desarrollo del trabajo, se enumeran las producciones científicas vinculadas a esta tesis y se realiza un breve resumen de la estructura de la misma.

En el segundo capítulo, *Estado de la cuestión*, se presentan los conceptos que son base para los objetivos de la tesis, sentando la base teórica para el desarrollo de la misma. Se presentan los

desarrollos existentes en el proceso unificado de desarrollo de software, en diagramas UML aplicados a dicho proceso y en métricas de software. Las métricas de software, núcleo de esta tesis, son desarrolladas en profundidad en este capítulo, describiéndose las diferencias entre métricas dinámicas y métricas estáticas y entre métricas a nivel de diseño y métricas a otros niveles del proceso de desarrollo. Se analizan decenas de métricas, incluyendo algunas específicas para diagramas UML y por último se describen los avances en la aplicación de conceptos de redes complejos a sistemas software.

En el tercer capítulo, *Descripción del problema*, se explica la importancia de la medición a lo largo del proceso de desarrollo de software, identificando el problema de investigación a partir de las posibilidades de mejora detectadas en dicha área. Se caracteriza el problema abierto que intenta solucionar esta tesis, finalizando con un sumario de investigación en el cual se plantean una serie de preguntas que se pretende responder mediante este trabajo de tesis.

En el cuarto capítulo, *Solución propuesta*, se propone una metodología de análisis del comportamiento de sistemas software, generada desde el punto de vista del diseño de sistemas, dentro del paradigma orientado a objetos y basada en redes complejas, en cumplimiento con los objetivos propuestos en esta tesis. Asimismo, se especifican las generalidades de la metodología propuesta y se describen en gran profundidad sus dos actividades principales.

En el quinto capítulo, *Validación*, se analiza y estudia el comportamiento de la solución propuesta en el capítulo anterior. Se realiza una introducción al método de validación aplicado, con las correspondientes justificaciones de su elección y se describe que partes de la metodología serán sometidas a estudio, presentando al final los resultados del mismo.

En el sexto capítulo, *Conclusiones*, se puntualizan las conclusiones obtenidas del desarrollo de esta tesis, respondiendo a las preguntas de investigación planteadas en el tercer capítulo. Se realiza una valoración del aporte de esta tesis al análisis dinámico de los sistemas y se identifican futuras líneas de trabajo en esta área.

En el séptimo capítulo, *Referencias*, se listan en orden alfabético todas las referencias bibliográficas utilizadas para el desarrollo de esta tesis.

2. ESTADO DE LA CUESTIÓN

En este capítulo se presenta el estado de la cuestión de los conceptos que son base para los objetivos de la presente tesis. Se presentan los desarrollos existentes en el proceso unificado de desarrollo de software (sección 2.1), en los diagramas UML para el flujo de análisis y diseño de RUP (sección 2.2) y en métricas de software (sección 2.3), ampliándose a continuación los conceptos referentes a las métricas dinámicas y sus ventajas sobre las estáticas (sección 2.4) y a las métricas a nivel de diseño, comparándolas con las métricas existentes en las demás etapas del desarrollo de sistemas (sección 2.5). Se desarrolla además una sección exclusiva de métricas específicas para diagramas UML (sección 2.6) y finalmente los conceptos que permiten la aplicación de la teoría de redes complejas al modelado y la medición de sistemas de información (sección 2.7).

2.1 Proceso unificado de desarrollo de software

El proceso unificado de desarrollo de software [Jacobson, Booch y Rumbaugh. 1999] es un proceso de desarrollo de software iterativo e incremental cuya implementación más popular se denomina RUP (*Rational Unified Process* o Proceso Unificado de Rational). El proceso unificado de desarrollo de software, o más simplemente, proceso unificado, es un marco extensible de desarrollo que puede ser personalizado según las necesidades de cada proyecto de desarrollo de software [Rational Unified Process, 2001].

Las principales características del proceso unificado son las siguientes:

- Es iterativo e incremental: el proceso unificado consta de cuatro fases (Inicio, Elaboración, Construcción y Transición) que se dividen en iteraciones y que entregan como resultado un incremento del producto que añade o mejora las funcionalidades existentes. Estas cuatro fases representan el ciclo de vida del proyecto de desarrollo.
- Está dirigido por casos de uso: los casos de uso, parte del Lenguaje Unificado de Modelado (descripto en la sección 2.2), se utilizan para capturar los requisitos funcionales y definir, en función de ellos, los contenidos de las iteraciones.
- Está centrado en la arquitectura: el proceso unificado considera que la arquitectura es clave para el desarrollo del sistema ya que la misma debe dar soporte a los casos de uso que dirigen el desarrollo. A tal fin, propone el uso de diversos modelos de arquitectura para representar los distintos aspectos del sistema a desarrollar.
- Está enfocado en los riesgos: el proceso unificado requiere que el equipo de proyecto identifique los riesgos críticos para poder considerarlos primeramente en cada iteración.

Dentro de cada una de las fases del ciclo de vida, las actividades se dividen en flujos de trabajo o disciplinas, tal como puede observarse en la figura 2.1. Esta figura de dos ejes permite describir al proceso considerando el eje horizontal como el tiempo, en donde se puede observar el aspecto dinámico del proceso a medida que se va ejecutando y en donde el mismo es expresado en términos de ciclos, fases, iteraciones e hitos. A su vez, el eje vertical representa el aspecto estático (o estructura) del proceso, es decir, la forma en que se describe el mismo en términos de actividades, artefactos, roles y flujos de trabajo [Rational Unified Process, 2001].

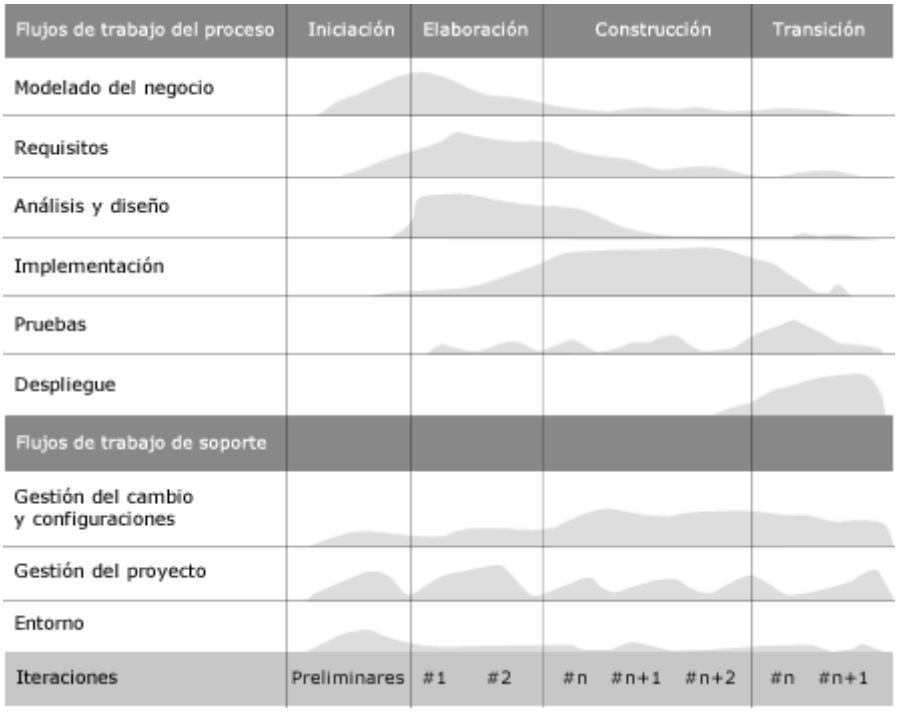


Figura 2.1 – Ciclo de vida del proceso unificado de Rational

El ciclo de vida del proceso unificado de desarrollo de software está dividido en ciclos siendo cada ciclo una nueva generación del producto software. Cada ciclo, eje horizontal de la figura 2.1, está dividido a su vez en las cuatro fases mencionadas previamente (Iniciación, Elaboración, Construcción y Transición), cada una de las cuales finaliza con un hito definido. Este hito implica el cumplimiento de una serie de objetivos.

El objeto de esta tesis está circunscripto completamente dentro de las fases de Elaboración y Construcción. La fase de Elaboración tiene como objetivo analizar el problema de dominio, establecer una sólida base arquitectónica, desarrollar el plan del proyecto y eliminar los elementos de mayor riesgo del proyecto. La fase de Construcción, a su vez, tiene como objetivo desarrollar todos los componentes y las características del software e integrarlas en un único producto.

La estructura del proceso, o eje vertical de la figura 2.1, describe quién realiza qué tareas, cuándo y cómo. El quién es representado por los roles, el qué por los artefactos, el cuándo por los flujos de trabajo y el cómo por las actividades.

El proceso unificado define a los flujos de trabajo como la secuencia de actividades que producen un resultado de valor, mostrando todos los roles, actividades y artefactos involucrados junto con las relaciones entre ellos. Cada rol, cumplido por una persona o un equipo de trabajo, ejecuta una serie de actividades y es dueña de una serie de artefactos. Las actividades son unidades de trabajo, llevadas a cabo por los roles, que normalmente tienen como fin crear o actualizar algún artefacto. Los artefactos, a su vez, son productos tangibles del proyecto que poseen información la cual es producida, modificada o usada por un proceso. A modo de ejemplo se puede identificar el rol de Arquitecto de Sistemas que desempeña la actividad de Análisis de Casos de Uso en la cual se genera el artefacto Diagrama de Casos de Uso.

El Diseño de Sistemas, objeto de estudio de la presente tesis, se encuentra dentro del flujo de trabajo Análisis y Diseño, que se desarrolla principalmente en la fase de Elaboración y es el que determina como se van transformar los requisitos en un producto software. El resultado final del flujo de trabajo de análisis y diseño es el artefacto Modelo de Diseño, siendo opcional realizar el artefacto Modelo de Análisis. El Modelo de Diseño representa una abstracción del código fuente en la cual se modelan clases, interfaces y la relación entre ellas, además de otros aspectos. Las actividades de este flujo de trabajo están centradas en el concepto de arquitectura, una de las principales características del proceso unificado mencionada previamente en esta sección. La arquitectura se representa mediante una serie de vistas de arquitectura que son abstracciones del diseño completo del sistema en las cuales ciertas características se hacen más visibles que otras.

La transformación de requisitos llevada a cabo en el flujo de trabajo Análisis y Diseño se realiza mediante un conjunto de artefactos tales como un modelo de casos de uso o un diagrama de arquitectura. Si bien el análisis y el diseño están considerados por el proceso unificado de Rational como un único flujo de trabajo, son actividades diferentes con artefactos diferentes. En la figura 2.2 puede observarse el flujo de trabajo de RUP para el análisis y diseño. Cabe considerar que al ser el proceso unificado de Rational iterativo e incremental, el flujo de trabajo que puede observarse corresponde a una única iteración del proyecto.

Dentro del flujo de trabajo de Análisis y Diseño se utiliza un conjunto de artefactos que van evolucionando con las iteraciones que se realizan sobre el proyecto. Tal como se describió previamente en esta sección, siendo el proceso unificado un marco extensible de desarrollo que puede

ser personalizado, los artefactos a utilizar dependerán de las características del proyecto y de los responsables del desarrollo del mismo.

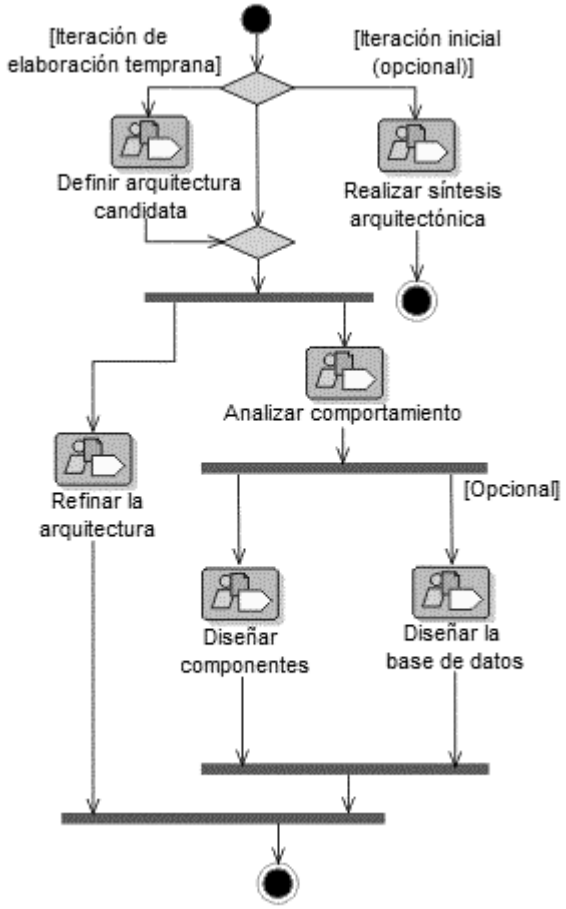


Figura 2.2 – Flujo de trabajo de RUP para el Análisis y Diseño

2.2 Diagramas UML para el flujo de análisis y diseño de RUP

El lenguaje unificado de modelado, o UML (*Unified Modeling Language*), es un lenguaje multipropósito de modelado de sistemas que fue diseñado como un estándar para la visualización del diseño de sistemas software [Booch *et al.*, 2005]. El lenguaje fue adoptado en 1997 como un estándar por parte de la organización OMG (*Object Management Group*) y en el año 2000 por parte de la ISO (*International Organization for Standardization* u Organización Internacional para la estandarización). Ha sido aceptado como estándar por la mayoría de empresas de desarrollo de software [Genero et al., 2005].

Dentro de UML se manejan dos conceptos esenciales, el concepto de modelo y el de diagrama. El modelo es la representación abstracta del sistema, representada gráficamente mediante un diagrama. Cada diagrama representa de forma parcial al modelo y no necesariamente todo el modelo se encuentra representado mediante diagramas, ya que de forma análoga a RUP, la aplicación de UML puede ser personalizada, elaborando sólo los diagramas que se consideren necesarios.

Otro concepto esencial de UML es el de vista. Una vista es un subconjunto de construcciones de modelado que se enfocan en un aspecto particular del sistema [Torossi, 2005]. Los diagramas UML pueden representar tres vistas diferentes del modelo del sistema. Una vista estática o estructural, que describe el sistema mediante el uso de objetos, atributos, operaciones y relaciones; una vista dinámica o de comportamiento que enfatiza el comportamiento del sistema mostrando las colaboraciones entre los objetos y los cambios que van sufriendo los mismos en el tiempo; y una vista de gestión que describe la organización de los modelos y diagramas en unidades jerárquicas.

El principal factor de diferencia entre la vista estructural y la vista de comportamiento es el tiempo, ya que este no es considerado en la primera y es la esencia de la segunda. De hecho, podría considerarse a los diagramas de comportamiento como una descripción de los cambios que se producen en los diagramas de vista estructural a lo largo del tiempo.

A modo de ejemplos, la vista estática o estructural, incluye los diagramas de clase y los diagramas de estructura mientras que la vista dinámica o de comportamiento, incluye los diagramas de secuencia, los diagramas de actividad y los diagramas de máquina de estado. En la figura 2.3 se pueden observar los principales diagramas de comportamiento de UML.

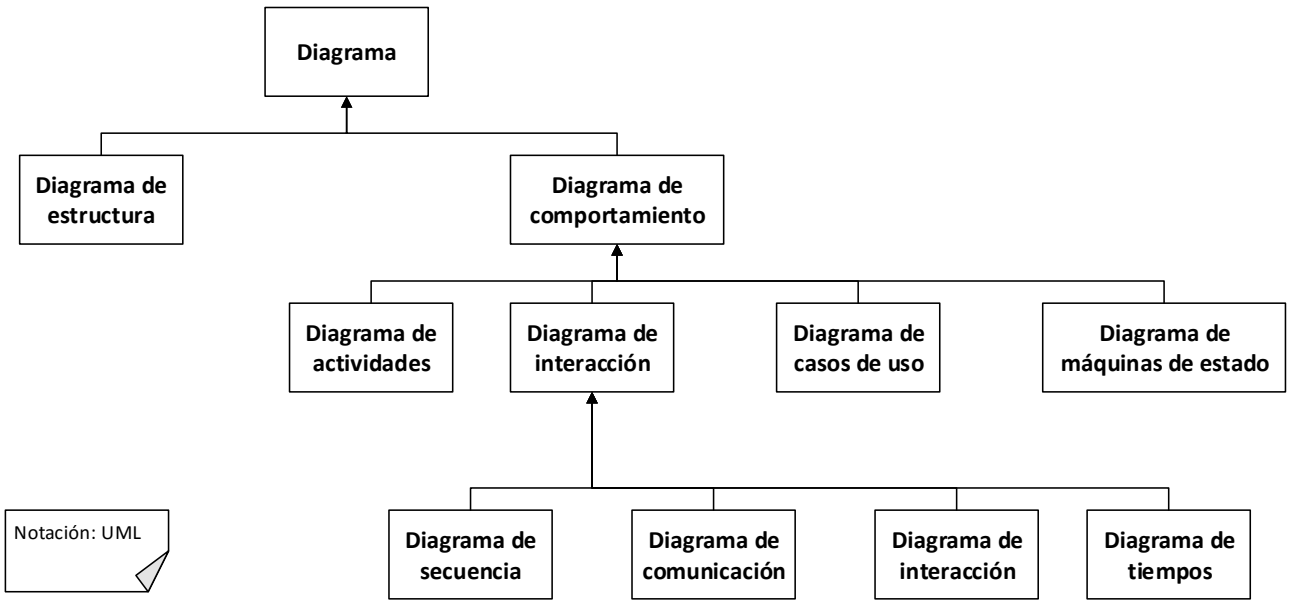


Figura 2.3 – Diagramas de comportamiento UML v2.x

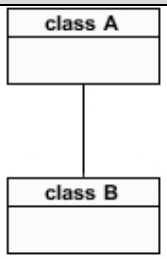
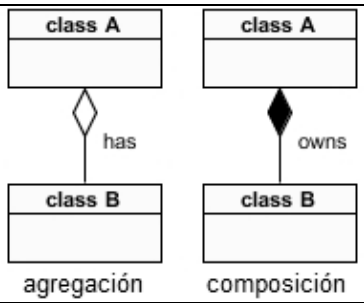
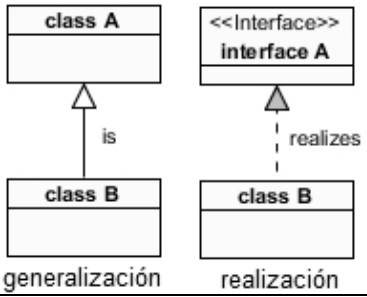
UML provee todos los diagramas necesarios para la elaboración de los artefactos requeridos en el flujo de trabajo de análisis y diseño de RUP. Dentro de este flujo de trabajo, la actividad de diseño está soportada por varios diagramas, o artefactos, según terminología de RUP. Dos de los principales son el diagrama de clases y el diagrama de secuencia, clasificado como diagrama de interacción.

El diagrama de clases describe la estructura del sistema mostrando las clases del mismo, enfocándose en sus atributos, clases y relaciones. Los diagramas de interacción, por su parte, se enfocan en el flujo de datos y control entre los componentes del sistema que se está modelando. En particular, el diagrama de secuencia modela el flujo o secuencia de mensajes que se envían los objetos entre sí.

El diagrama de clases es la vista interna más importante de un sistema ya que es la base sobre la cual se desarrolla todo el trabajo posterior [Genero *et al.*, 2002]. Dentro de un diagrama de clases, las clases se representan como rectángulos con tres divisiones, anotando en la primera división el nombre de la clase, en la segunda los atributos de la misma y en la tercera sus métodos.

Tanto los atributos como los métodos tienen propiedades dentro de UML. Para el caso de los atributos, por ejemplo, se definen propiedades como la visibilidad (público, protegido o privado) y la posibilidad de modificarse (atributo de solo lectura). Los métodos, u operaciones, comparten algunas de estas propiedades.

Las principales relaciones entre clases tienen su correspondiente notación de modelado en UML. En la tabla 2.2 se detallan los tipos de relación más importantes entre clases y su correspondiente modelado en UML v2.4.1 [OMG, 2011].

Relación	Notación	Descripción
Asociación		La asociación es una conexión semántica entre dos o más clases y proporciona una conexión para el envío de mensajes. Permite la interacción de clases en un sistema.
Agregación y Composición		La agregación es una relación en la cual los objetos de una clase están formados por objetos de otra(s) clase(s). La composición es una agregación con dependencia existencial entre las clases agregadas.
Generalización y realización		La generalización es una relación en la cual una clase es una especialización de otra. En la realización una clase implementa (o ejecuta) la funcionalidad de otra, normalmente llamada interface.

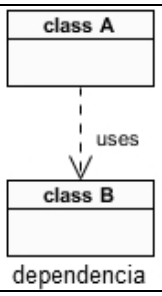
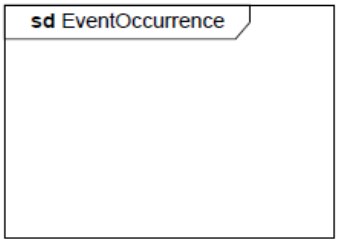
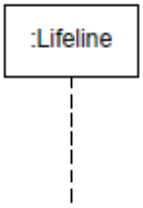
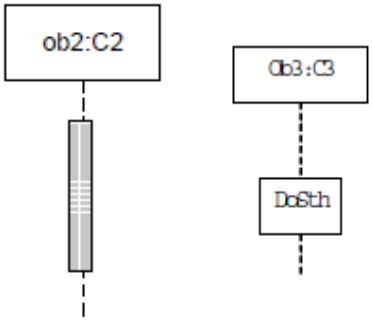
Dependencia		La dependencia implica una relación de uso, en la que una clase hace uso de un método o atributo de otra. Es la relación más débil.

Tabla 2.1 – Principales relaciones entre clases y su modelado en UML v2.4.1 [OMG, 2011]

El diagrama de secuencia es el tipo de diagrama de interacción más común y se enfoca en el intercambio de mensajes entre un conjunto de líneas de vida [OMG, 2011]. Este diagrama se representa mediante un gráfico de dos dimensiones en el cual se modela el tiempo en el eje vertical y los objetos (identificados como roles) en el eje horizontal. De cada objeto nace una línea vertical que representa el estado del objeto siendo que la línea en si representa la vida del mismo. Entre estas líneas verticales se trazan otras líneas horizontales las cuales representan los mensajes que se envían los objetos entre sí.

Existen diversos nodos a utilizar en el diagrama de secuencia, que evolucionan con las diferentes versiones de la superestructura de UML, definida por el OMG. En la tabla 2.1 se pueden observar los principales nodos especificados en la superestructura de UML v2.4.1 [OMG, 2011].

Tipo de nodo	Notación	Descripción
Marco (*Frame*)		Es un marco rectangular dentro del cual se desarrolla la interacción. Posee un nombre en la esquina superior izquierda a modo de referencia.
Línea de vida		La línea de vida representa la existencia de un objeto. Mientras se encuentre punteada el objeto existirá pero no se encontrará en ejecución.
Ejecución		La ejecución, que posee dos representaciones posibles, se representa normalmente mediante una doble línea las cuales indican el inicio y el fin de la ejecución del objeto.
Destrucción		La destrucción, o muerte, de un objeto se representa mediante una cruz sobre la línea de vida.

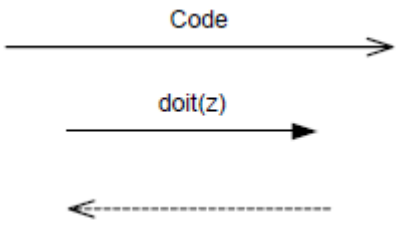
Mensaje		Existen diferentes tipos de mensaje entre los objetos. Dentro de los mensajes completos existen los mensajes asincrónicos, las llamadas y las respuestas, indicadas en la columna anterior en este mismo orden.

Tabla 2.2 – Principales nodos de UML v2.4.1 [OMG, 2011]

2.3 Métricas de software

Dentro del campo de la Ingeniería del Software, se define como métrica a una medida o conjunto de medidas cuyo fin es cuantificar alguna característica de un sistema software, o parte del mismo, con el objetivo de entenderlo y administrarlo a lo largo de todo su ciclo de vida [Chhabra y Gupta, 2010; Pressman, 2005; IEEE 1990].

El primer libro dedicado a las métricas de software fue publicado en el año 1976 [Gilb, 1976], aunque los primeros avances en la materia corresponden a mediados de la década de 1960 con el surgimiento de la métrica de líneas de código [Fenton y Neil, 2000].

La mayoría de los autores que realizaron estudios globales del estado de las métricas de software [Kitchenham, 2009; Bellini *et* al, 2008; Kaner, 2004; Fenton y Neil, 2000; Glass, 1994] coincide en que a pesar de ser un área con varias décadas de desarrollo, las métricas de software todavía no se encuentran en un nivel de madurez adecuado para una disciplina sistémica y cuantificable como la Ingeniería del Software. El principal motivo por el cual se da esta situación, y que lleva al relativo fracaso de la mayoría de los programas de aplicación de métricas, es que las mismas no proveen la información necesaria para la toma de decisiones de gestión durante el ciclo de vida del desarrollo de software [Fenton y Neil, 2000].

Si bien las métricas de software todavía no poseen terminología, principios o métodos consolidados, de una forma general se puede afirmar que permiten obtener una mejor visibilidad y compresión del software estableciendo las bases para mejoras además de permitir el planeamiento, monitoreo y control de procesos, productos y recursos involucrados en los procesos de desarrollo de software [Bellini *et al*, 2008].

El término métricas de software engloba a un amplio y variado grupo de actividades y métricas dentro de la Ingeniería del Software que permiten estudiar desde características intrínsecas del producto software, tanto cualitativas como cuantitativas, hasta estimar los recursos necesarios para la construcción del mismo [Fenton y Neil, 2000]. En este punto es importante diferenciar el término métrica de los términos medida y medición. De acuerdo a Pressman [Pressman, 2005], estos

conceptos presentan sutiles diferencias ya que mientras una medida representa una indicación cuantitativa de algún atributo del sistema y la medición es el acto de determinar esta medida, la métrica es la medida cuantitativa del grado en el cual el sistema posee un atributo determinado [ANSI/IEEE, 2007].

El objetivo del cálculo de las métricas es el de obtener un indicador, el cual permite obtener conocimiento acerca del sistema en estudio y a partir del cual se puedan tomar las decisiones correspondientes. Este indicador apunta a la medición de la calidad de un sistema software, calidad que puede ser definida a través de atributos externos del software como la funcionalidad, la reusabilidad, la usabilidad, la eficiencia, el mantenimiento y la portabilidad [ISO, 2001] o a través de atributos internos como la complejidad, el acoplamiento y la cohesión, entre otros.

Las métricas de software, tal como son vista en esta tesis, se refieren exclusivamente a las métricas de producto que se concentran en atributos específicos de los productos de trabajo de la ingeniería del software y se recopilan a medida que se realizan las tareas técnicas [Pressman, 2005]. Existen otras dos categorías de métricas que son las métricas de proceso y las métricas de proyecto. Las métricas de proceso son aquellas que se recopilan a lo largo de diversos proyectos y tienen un impacto a largo plazo. Son métricas aplicadas a fines estratégicos que guían avances en el proceso de desarrollo de software y en el ambiente en el cual se desarrolla el mismo. Las métricas de proceso se aplican sobre todos los proyectos de desarrollo de software durante un largo período de tiempo. Las métricas de proyecto, en cambio, son aquellas que permiten a un gestor de proyecto determinar el estado de un proyecto de software [Basso, 2014; Pressman, 2005]. Estas métricas son tácticas y no tienen impacto en los demás proyectos de desarrollo de software en curso. Su duración está acotada al ciclo de vida de un proyecto de software específico y su aplicación permite realizar ajustes sobre el mismo.

Con el objetivo principal de eliminar o acotar algunos sesgos comunes a la hora de medir software, en particular la calidad del mismo, entre los años 1991 y 2004 se desarrolló el estándar ISO/IEC 9126 (*Software Product Quality* o Calidad del producto software) que proporciona un modelo para la evaluación de la calidad del software, estableciendo un lenguaje claro y normalizado para convertir definiciones abstractas en valores medibles. El estándar está dividido en 4 partes según se detalla a continuación:

- ISO/IEC 9126-1, en donde se presenta un modelo de calidad, clasificando a la calidad del software como una estructura de características y sub-características.
- ISO/IEC 9126-2, en donde se definen métricas externas, las cuales son aplicables al software en ejecución, o sea, desde un punto de vista dinámico.

- ISO/IEC 9126-3, en donde se definen métricas internas, que no dependen de la ejecución del software; es decir, en estado estático.
- ISO/IEC 9126-4, en donde se define un conjunto de métricas de uso, las cuales son aplicadas sobre el contexto en el cual se ejecuta el software.

Las métricas de las secciones 2, 3 y 4 del estándar ISO/IEC 9126 se aplican sobre las características y sub-características de la sección 1 del mismo. Su diferencia radica en que la sección 3 evalúa el software en sí mismo, mientras que la sección 2 evalúa el comportamiento del software y la sección 4 evalúa los efectos del software en un contexto de uso particular. La ISO indica explícitamente que las métricas definidas en este estándar no son exhaustivas, lo que significa que no todos los aspectos del software están cubiertos por el mismo.

El estándar ISO/IEC 9126 realiza además una clasificación de métricas según su naturaleza, catalogándolas como:

- Métricas básicas o directas, que son aquellas que se obtienen directamente del análisis de parte del software no involucrando ningún cálculo u otra métrica
- Métricas de agregación, que son aquellas métricas generadas a partir de la aplicación de cálculos sobre las métricas directas
- Métricas derivadas o indirectas, que son aquellas que están compuestas por una función que utiliza más de una métrica básica o de agregación como entrada a una función

El estándar ISO/IEC 9126, junto con el ISO/IEC 15598 (*Software Product Evaluation* o Evaluación del producto software), fue reemplazado en el año 2005 por el proyecto SQuaRE (*Software Product Quality Requirements and Evaluation* o Requerimientos y evaluación de la calidad del producto software), dando lugar al estándar ISO/IEC 25000. Este estándar, que fue revisado en el año 2014, dio lugar a una nueva versión del mismo y se encuentra orientado a la calidad del software como producto.

El propósito del estándar ISO/IEC 25000 es guiar el desarrollo del software mediante la especificación de requisitos y la evaluación de atributos de calidad. El estándar ISO/IEC 25000 se encuentra dividido en 5 familias de normas:

- ISO/IEC 2500n (*División de gestión de calidad*) en la cual se definen todos los modelos, términos y definiciones comunes referenciados por las demás normas de la familia ISO/IEC 25000. Actualmente incluye:

  - ISO/IEC 25000 (*Guía de SQuaRE*) que incluye la arquitectura de SQuaRE, la terminología, un resumen de las normas, las partes involucradas y los modelos de referencia
  - ISO/IEC 250001 (*Planeamiento y Gestión*) que establece los requisitos para gestionar la evaluación y especificación de los requisitos de software
- ISO/IEC 2501n (*División de modelo de calidad*) en la cual se presentan modelos de calidad incluyendo características para calidad interna, externa y de uso del producto software (de forma análoga a las ISO/IEC 9126). Incluye:
  - ISO/IEC 25010 (*Modelos de calidad del sistema y el software*) que describe el modelo de calidad para el producto software, detallando características y subcaracterísticas de calidad
  - ISO/IEC 25012 (*Modelo de calidad de datos*) que define un modelo para la calidad de datos
- ISO/IEC 2502n (*División de medición de calidad*) que incluyen un modelo de referencia de la medición de calidad del producto, incluyendo definiciones de mediciones de medida de calidad y guías prácticas para su aplicación. Actualmente está formada por:
  - ISO/IEC 25020 (*Modelo y guía de referencia de la medición*) que realiza una introducción al modelo de referencia común de medición de la calidad y proporciona una guía para que los usuarios seleccionen y/o desarrollen las medidas propuestas por la ISO
  - ISO/IEC 25021 (*Modelos de medición de la calidad*) presenta un conjunto de métricas básicas, agregadas y derivadas que pueden ser usadas a lo largo de todo el ciclo de vida del desarrollo del producto software
  - ISO/IEC 25022 (*Medición de la calidad en uso*) que define las métricas para la medición de la calidad en uso del producto software
  - ISO/IEC 25023 (*Medición de la calidad del sistema y el producto software*) que define métricas para realizar la medición de la calidad de sistemas y productos software
  - ISO/IEC 25024 (*Medición de la calidad de datos*) que define las métricas para la medición de la calidad de datos
- ISO/IEC 2503n (*División de requisitos de calidad*) familia de normas en la cual se desarrolla un conjunto de recomendaciones y guías para especificar requisitos de calidad que serán usados en el proceso de elicitación de requisitos de calidad del producto software a desarrollar. Actualmente incluye una única norma:
  - ISO/IEC 25030 (*Requisitos de calidad*) que provee un conjunto de sugerencias para elaborar la especificación de los requisitos de calidad del producto software

- ISO/IEC 2504n (*División de evaluación de calidad*) formada por normas que proporcionan requisitos, recomendaciones y guías para la evaluación del producto software. Incluye:
    - ISO/IEC 25040 (*Guía y modelo de referencia de evaluación*) que provee un modelo de referencia para la evaluación de la calidad
    - ISO/IEC 25041 (*Guía de evaluación para desarrolladores*)
    - ISO/IEC 25042 (*Módulos de evaluación*)
    - ISO/IEC 25045 (*Módulo de evaluación para la recuperabilidad*)
- ISO/IEC 25050 a 25099, reservadas para extensiones o informes técnicos que aborden dominios de aplicación específicos. Actualmente incluye las siguientes normas:
    - ISO/IEC 25051 (*Requerimientos para la calidad de Productos Software listos para usar (RUSP) e instrucciones para pruebas)*
    - ISO/IEC 25062 (*Formato común industrial (CIF) para reportes de pruebas de usabilidad*)

El desarrollo de métricas de software debe cumplir con una serie de principios que definan no sólo características deseables de las mismas sino que también determinen su aplicación. La aplicación de las métricas requiere de un proceso sistemático de medición. Segú Roche [Roche, 1994], las actividades de este proceso son:

1. Formulación: actividad en la cual se definen las métricas a aplicar, en función del software sobre el cual se aplican. Las métricas a aplicar pueden ser parte de un conjunto de métrica o derivación de las mismas.
2. Recolección: mecanismo mediante el cual se obtendrán los datos necesarios para la aplicación de las métricas. Roche sugiere que esta actividad sea automatizada dentro de lo posible.
3. Análisis: cálculo de métricas en base a los datos recopilados en la etapa anterior. Roche también sugiere que esta actividad sea automatizada.
4. Interpretación: es el estudio y evaluación del resultado del análisis. En esta actividad se busca definir las características del software medido en función a los resultados obtenidos de las métricas aplicadas. Roche sugiere que cada métrica posea directrices y recomendaciones para su interpretación.
5. Retroalimentación: esta actividad propone retroalimentar el proceso de desarrollo de software con los resultados interpretados de la aplicación de métricas.

Además de un proceso sistemático de medición, las métricas de software deben cumplir con una serie de características y principios. Según Pressman [Pressman, 2005] algunos de los principios a cumplir son los siguientes:

- *Una métrica debe tener propiedades matemáticas deseables*. Este principio indica que el valor de una métrica debe encontrarse dentro de un rango numérico significativo. Asimismo, la métrica no debería contar con componentes ordinales.
- *Una métrica debe tener el mismo sentido que la característica del software que mide*. Lo que significa que una métrica que mide una característica positiva del software debe aumentar a mayor presencia de esta característica y disminuir en caso contrario.
- *Cada métrica debe validarse.* Es importante que las métricas sean validadas en diversos contextos antes de su aplicación en escenarios reales para la toma de decisiones.

De forma más específica Ejiogu [Ejiogu, 1991] desarrolla un conjunto de atributos que cada métrica de software debe cumplir. Si bien esta serie de atributos fue desarrollada específicamente para las métricas derivadas, los mismos también son válidos para las métricas directas y de agregación. Los atributos son:

- *Simples y calculables*: el cálculo de las métricas no debe exigir grandes esfuerzos y su comprensión debe ser relativamente sencilla.
- *Empírica e intuitivamente persuasivas*: las métricas deben poder asociarse a los conceptos y nociones del ingeniero de forma tal de que pueda asociarlas fácilmente al atributo o característica del software que está midiendo
- *Consistentes y objetivas*: las métricas no deben arrojar resultados ambiguos
- *Consistentes en el uso de unidades y dimensiones*: el cálculo, uso y la combinación de métricas deben mostrar uniformidad en el uso de unidades y dimensiones
- *Independientes del lenguaje de programación*: las métricas deben basarse en modelos de análisis o diseño o en la propia estructura del programa
- *Mecanismos efectivos para la retroalimentación de alta calidad*: debe existir un mecanismo de retroalimentación que efectivamente permita mejorar el producto software

Además de las características mencionadas previamente, las métricas de software deben cumplir con una serie de criterios de validación. El estándar IEEE 1061 [Kaner y Bond, 2004; IEEE, 1998] define los siguientes:

- *Correlación:* la métrica debe estar relacionada linealmente con el atributo de software medido.
- *Consistencia:* sea F el atributo medible del software e Y la salida de la aplicación de función métrica M sobre dicho atributo. M debe ser una función monótona. Es decir que si $f_1 > f_2 > f_3$ entonces $y_1 > y_2 > y_3$
- *Seguimiento:* para la función de la métrica M: F -> Y, a medida que F varía entre f_1 y f_2, Y debe variar entre y_1 e y_2 proporcionalmente

- *Previsibilidad:* para la función de la métrica M: F -> Y, si conocemos el valor de Y en determinado punto del tiempo, deberíamos poder predecir el valor de F
- *Diferenciación:* una métrica debe poder diferenciar entre componentes de software de alta calidad y componentes de software de baja calidad. El valor de las métricas debe permitir apreciar claramente esta diferencia.
- *Confiabilidad*: la métrica debe demostrar los cinco criterios de validación enunciados previamente para al menos un valor p del universo de aplicación de la métrica

En la literatura existente de métricas se ha propuesto una amplia variedad de taxonomías [Pressman, 2005]. Sin embargo, la mayoría de los autores considera principalmente dos clasificaciones: una primera clasificación en función del estado del software, que clasifica a las métricas en estáticas y dinámicas, y una segunda clasificación relativa a la etapa del proceso de desarrollo de software en el cual se aplica la métrica, que permite clasificar las métricas en métricas para el modelo de análisis, métricas para el modelo de diseño, métricas para el código fuente y métricas para pruebas. En la sección 2.4 se analizará la primera de estas clasificaciones mientras que en la sección 2.5 se analizará la segunda.

2.4 Métricas dinámicas

En esta sección se definen las métricas dinámicas y se describen sus principales diferencias con las métricas estáticas (sección 2.4.1). Posteriormente se enumeran y explican las métricas dinámicas más importantes dentro de la ingeniería del software, agrupándolas por característica de análisis y considerando el acoplamiento (sección 2.4.2), la cohesión (sección 2.4.3), la complejidad (sección 2.4.4) y el polimorfismo (sección 2.4.5). En la sección 2.4.6 se describen otras métricas dinámicas no englobadas en la agrupación anterior y, por último, en la sección 2.4.7 se introduce el concepto de métricas pseudodinámicas.

2.4.1 Definición de métricas dinámicas y diferencias con métricas estáticas

En función del estado del software al momento de la aplicación de las métricas, estas pueden clasificarse en estáticas y en dinámicas. Las métricas estáticas son aquellas que estudian qué puede suceder cuando un programa es ejecutado y, por su parte, las métricas dinámicas estudian efectivamente qué sucede en la ejecución del mismo [Kaur *et al.*, 2009].

Las métricas estáticas se concentran en propiedades estáticas del software y han sido hasta la actualidad las métricas con mayor presencia en la literatura de la materia [Tahir y MacDonell, 2012]. Estas métricas permiten cuantificar varias características del software, pero su habilidad de determinar el comportamiento de una aplicación todavía no ha sido demostrado [Chhabra y Gupta,

2010]. La limitación de las métricas estáticas para evaluar el comportamiento del software se hace evidente dentro del paradigma orientado a objetos, ya que algunas de sus características, como el uso de clases y el polimorfismo, solo pueden ser analizadas en ejecución.

Las métricas dinámicas, en cambio, capturan el comportamiento del software y son obtenidas a partir del análisis de la ejecución de parte del software o de algún modelo en un escenario particular, análisis que forma parte del paradigma de análisis dinámico [Cornelissen, 2009; Ball, 1999]. Las métricas dinámicas han demostrado proveer información más precisa sobre el funcionamiento del software en diversos estudios [Chhabra y Gupta, 2010; Quynh y Thang, 2009; Rothlisberger *et al.*, 2009].

Además de proveer información más precisa sobre el funcionamiento del software, las métricas dinámicas presentan otras diferencias con respecto a las métricas estáticas. En la tabla 2.2 se analizan las mismas:

Métricas estáticas	Métricas dinámicas
Fáciles de obtener	Difíciles de obtener
Disponibles en las primeras etapas de desarrollo	Disponibles en las etapas finales de desarrollo
Menos precisas en atributos cualitativos	Más precisas en atributos cualitativos
Centradas en la estructura del software	Centradas con el comportamiento del software
Ineficientes en el análisis de atributos del paradigma OO o código sin uso	Eficientes para el análisis de atributos del paradigma OO y código sin uso
Menos precisas para sistemas en tiempo real	Más precisas para sistemas en tiempo real

Tabla 2.3 – Comparación entre métricas estáticas y métricas dinámicas [Chhabra y Gupta, 2010]

La aplicación y el uso de métricas estáticas es más sencillo que el uso de métricas dinámicas ya que para su aplicación no es necesaria la ejecución de software. Además, las métricas estáticas pueden aplicarse prácticamente en cualquier etapa del desarrollo de software, incluyendo las etapas iniciales. La aplicación de métricas dinámicas, en cambio, requiere de la ejecución de código o de modelos de ejecución, los cuales normalmente son elaborados en las etapas finales del ciclo de vida del desarrollo de software. Sin embargo, los beneficios intrínsecos de las métricas dinámicas (como su mayor precisión y su eficiencia en el paradigma orientado a objetos), compensan con creces su mayor dificultad de obtención o la necesidad de tener que esperar a etapas más avanzadas del desarrollo de software [Chhabra y Gupta, 2010].

La obtención de datos para el cálculo de métricas dinámicas puede obtenerse a partir de trazas de ejecución, mediante la aplicación del paradigma de análisis dinámico de software, o a través de la simulación de software, mediante UML o ROOM. El análisis de trazas de ejecución tiene como ventaja una mayor precisión pero tiene la desventaja de solo ser posible de realizar en las etapas finales de desarrollo de software. La simulación, en cambio, provee resultados menos precisos pero está disponible previamente [Tahir y MacDonell, 2012].

La mayor precisión de las métricas dinámicas con respecto a las estáticas se hace evidente en la medición de atributos cualitativos. Determinados factores como la calidad, la confiabilidad y la reusabilidad difícilmente se pueden obtener a partir de modelos o código estático ya que dependen del comportamiento del software en ejecución. Es en la ejecución del código o de modelos de ejecución en donde se pueden obtener atributos de calidad como la performance y el ratio de errores.

Otra diferencia importante, relacionada con la capacidad de las métricas dinámicas de analizar no sólo la estructura del sistema sino también su comportamiento, es que las métricas dinámicas permiten realizar mediciones de atributos del paradigma orientado a objetos que las métricas estáticas no. La naturaleza de este paradigma, con características como la herencia, el polimorfismo y el *late binding*, hace que no sea eficiente el realizar un análisis estático del software.

Si bien, y tal como se indicó previamente en esta sección, las métricas dinámicas tienen un menor desarrollo, existen decenas de métricas dinámicas propuestas en la literatura. A continuación se describen varias de ellas [Tahir y MacDonell, 2012; Chhabra y Gupta, 2010].

2.4.2 Métricas dinámicas de acoplamiento

Se define como acoplamiento al grado de interdependencia que hay entre distintas partes de un programa. Es uno de los conceptos más importantes del diseño de sistemas. En general se busca que sea mínimo ya que las partes (módulos o clases) de un programa deberían ser independientes y un cambio en una de ellas no debería afectar a las demás. Las métricas dinámicas de acoplamiento son las más estudiadas hasta el momento [Tahir y MacDonell, 2012]. A continuación se analizan las más representativas.

2.4.2.1 Métricas EOC e IOC

Desarrolladas por Yacoub [Yacoub *et al.*, 1999], las métricas EOC (*Export Object Coupling* o Acoplamiento de salida entre objetos) e IOC (*Import Object Coupling* o Acoplamiento de entrada entre objetos) son métricas que se aplican sobre modelos de ejecución orientados a objeto, que son generados mediante el lenguaje de modelado ROOM (Real-Time Object Oriented Modeling). Estos modelos permiten simular la ejecución de un programa, o parte del mismo, y a partir de dicha ejecución contar el número de mensajes enviados entre dos objetos cualesquiera. El resultado de cada métrica es un porcentaje que refleja la participación de un objeto en la ejecución de del programa, o la parte del mismo que se encuentra en análisis.

De forma numérica:

$$EOC_x(o_i, o_j) = \frac{\left|\{E_x(o_i, o_j) \mid (o_i, o_j) \in O \ \wedge \ o_i \neq o_j\right|}{MT_x} * 100$$

$$IOC_x(o_i, o_j) = \frac{\left|\{I_x(o_i, o_j) \mid (o_i, o_j) \in O \ \wedge \ o_i \neq o_j\right|}{MT_x} * 100$$

En donde:

- $E_x(o_i, o_j)$ es el número total de mensajes que salen de o_i hacia o_j
- $I_x(o_i, o_j)$ es el número total de mensajes recibidos por o_i desde o_j
- MT_x es el número total de mensajes intercambiados en la ejecución del modelo

2.4.2.2 Métricas de Arisholm

Arisholm [Arisholm *et al.*, 2004] extiende el concepto de las métricas de Yacoub para tener en consideración la dirección (*Import* o *Export*), el nivel de mapeo (clase u objeto) y la asociación o intensidad de la relación entre ambos objetos, la cual puede clasificarse en:

- Mensajes dinámicos (D): cantidad de veces que un mensaje se envía de un objeto a otro
- Métodos de invocación distintos (M): cantidad de métodos distintos que son invocados entre dos objetos
- Clases distintas (C): número de clases distintas involucradas en la asociación entre dos objetos

A partir de estas características, Arisholm da lugar a un conjunto de 12 métricas, las cuales pueden verse en la tabla 3.3. El conjunto de métricas de Arisholm fue validado teórica y empíricamente [Arisholm *et al.*, 2004].

Nombre de la métrica	Dirección del acoplamiento	Nivel de mapeo	Intensidad del acoplamiento
IC_OD	*Import Coupling* (IC)	Objeto (O)	Mensajes dinámicos (D)
IC_OM	*Import Coupling* (IC)	Objeto (O)	Métodos distintos (M)
IC_OC	*Import Coupling* (IC)	Objeto (O)	Clases distintas (C)
IC_CD	*Import Coupling* (IC)	Clase (C)	Mensajes dinámicos (D)
IC_CM	*Import Coupling* (IC)	Clase (C)	Métodos distintos (M)
IC_CC	*Import Coupling* (IC)	Clase (C)	Clases distintas (C)
EC_OD	*Export Coupling* (EC)	Objeto (O)	Mensajes dinámicos (D)
EC_OM	*Export Coupling* (EC)	Objeto (O)	Métodos distintos (M)
EC_OC	*Export Coupling* (EC)	Objeto (O)	Clases distintas (C)
EC_CD	*Export Coupling* (EC)	Clase (C)	Mensajes dinámicos (D)
EC_CM	*Export Coupling* (EC)	Clase (C)	Métodos distintos (M)
EC_CC	*Export Coupling* (EC)	Clase (C)	Clases distintas (C)

Tabla 2.4 – Métricas dinámicas de acoplamiento de Arisholm

A continuación se detalla el atributo de software que miden las métricas de Arisholm:

- IC_OD: número total de mensajes enviados desde un objeto al resto de los objetos
- IC_OM: número total de métodos distintos invocados por un objeto en los demás objetos

- IC_OC: número total de clases servidoras distintas usadas por los métodos de un objeto
- IC_CD: número total de mensajes enviados por todos los objetos de una clase
- IC_CM: número total de métodos distintos invocados por todos los objetos de una clase
- IC_CC: número total de clases servidoras distintas usadas por todos los objetos de una clase
- EC_OD: número total de mensajes recibidos por un objeto de todos los demás objetos
- EC_OM: número total de métodos recibidos por un objeto de los demás objetos
- EC_OC: número total de clases clientes usadas por un objeto
- EC_CD: número total de mensajes recibidos por todos los objetos de una clase
- EC_CM: número total métodos distintos recibidos por todos los objetos de una clase
- EC_CC: número total de clases clientes distintas usadas por todos los objetos de una clase

2.4.2.3 Métricas de Mitchell y Power

Las métricas de Mitchell y Power [Mitchell y Power, 2005] analizan el acoplamiento de entrada y de salida, al igual que las métricas de Arisholm, pero se concentran en el grado de las mismas más que en su nivel. Mitchell y Power proponen siete métricas, de las cuales tres trabajan sobre el concepto de acoplamiento estático y buscan determinar el acoplamiento de clases. Las restantes cuatro trabajan sobre el acoplamiento entre objetos. A continuación se analizan estas métricas:

- CBO dinámico para una clase: se obtiene contabilizando la cantidad de acoplamientos (acceso a métodos o variables de clase) de una clase a otras clases durante la ejecución del programa. Esta métrica está basada en la métrica CBO (*Coupling Between Object Classes*), definida por Chidamber y Kemerer [Chidamber y Kemerer, 1991], pero a diferencia de esta se aplica durante la ejecución del sistema.
- Grado de acoplamiento dinámico entre dos clases: define la cantidad de acoplamientos entre dos clases determinadas como porcentaje de la cantidad de acoplamientos de una de ellas en total.
- Grado de acoplamiento dinámico entre un conjunto de clases: es la misma métrica anterior pero aplicada a un conjunto de clases en lugar de a solo dos.
- R_I: acoplamiento de entrada entre objetos en ejecución. Se define como el número de clases al cual una clase accede para acceder a métodos o variables en ejecución.
- R_E: acoplamiento de salida entre objetos en ejecución. Se define como el número de clases que acceden a una determinada clase para acceder a métodos o variables en ejecución.
- RD_I: grado de acoplamiento de entrada. Se define como el R_I de una clase por sobre la sumatoria de los R_I de todas las clases.

- RD_E: grado de acoplamiento de salida. Se define como el R_E de una clase por sobre la sumatoria de los R_E de todas las clases.

2.4.2.4 Métrica dinámica de acoplamiento (DCM)

Hassoun *et al.* [Hassoun *et al.*, 2004] proponen un enfoque de acoplamiento diferente al anterior ya que se encuentra centrado en el concepto de tiempo. Este enfoque da lugar a la métrica dinámica de acoplamiento (DCM o *Dynamic Coupling Metric*) que se centra en la influencia de un objeto sobre otro, asumiendo diferentes estados para los objetos a lo largo de la ejecución del software. Esta métrica fue desarrollada para sistemas con arquitectura dirigida por modelos. De forma numérica su cálculo para un objeto está definido por:

$$DCM(P)|\Delta t = \sum_j \sum_i f_i(t_j) g_p(|O_i|)$$

Y para todo el sistema por:

$$DCM(sistema)|\Delta t = \sum_P DCM(P)$$

En donde:

- P es un objeto
- Δt es una secuencia ordenada de pasos de ejecución del programa
- i es el número de objetos acoplados al objeto en análisis P
- j es el número de pasos de ejecución del programa
- t_j representa a cada uno de los estados de ejecución
- $f_i(t_j)$ representa 1 o 0 dependiendo si existe acoplamiento o no en el estado j
- g_p representa la complejidad del objeto O acoplado a P
- $DCM(P)|\Delta t$ representa la sumatoria de todos los estados (y por ende, acoplamientos) entre el objeto P y el resto de los objetos del sistema (O_i) con los cuales interactúa P a lo largo de la ejecución del programa

2.4.2.5 Comparativa de métricas dinámicas de acoplamiento

Si bien las cuatro métricas descriptas previamente analizan el acoplamiento en ejecución, cada una de ellas lo hace de una forma distinta y con un enfoque particular. Las métricas EOC e IOC, desarrolladas por Yacoub, tienen la particularidad de poder aplicarse a partir de un modelo de diseño (ROOM), que se encuentra disponible en una etapa previa a la codificación, -etapa en la cual se

aplican el resto de las métricas analizadas-. Esta disponibilidad previa repercute en la precisión, la cual se hace mayor en la ejecución real del software.

Las métricas de Arisholm presentan un mayor nivel de detalle que las EOC e IOC debido a que además de la salida y la entrada utilizan otros conceptos como mensajes, métodos y clases. Estás métricas, sin embargo, son absolutas y no presentan una relación con el sistema total tal como lo hacen EOC e IOC y las métricas de Mitchell y Power.

2.4.3 Métricas dinámicas de cohesión

Se define como cohesión a la relación funcional existente entre las diversas partes (variables y métodos) de un módulo o clase. Normalmente se busca que sea alta ya que se desea que todas las partes de una clase estén relacionadas con la misma de una forma coherente. Es un atributo muy estudiado en el universo de métricas dinámicas [Tahir y MacDonell, 2012]. A continuación se detallan las más relevantes:

2.4.3.1 Métricas de Gupta y Rao

Gupta y Rao [Gupta y Rao, 2001] proponen nuevas definiciones para la cohesión funcional fuerte (SFC o *Strong Functional Cohesion*) y la cohesión funcional débil (WFC o *Weak Functional Cohesion*) sobre las cuales se toma normalmente la definición de Ott *et al.* [Ott *et al.*, 1995]. Los autores argumentan que para una medición dinámica de la cohesión se debe redefinir estos conceptos dado que la medición estática de la cohesión, mediante rebanamiento estático, sobreestima los valores reales de ésta. La redefinición de Gupta y Rao está basada en el concepto de rebanamiento dinámico [Page-Jones, 1988], una técnica usada para identificar todo el código de un programa en ejecución que pueda afectar a una variable determinada.

A partir de este concepto, Gupta y Rao definen métricas dinámicas basadas en los conceptos de definición común y uso común, considerando a la SFC como la cohesión modular para pares de definición y uso común de las rebanadas dinámicas para todas las variables de salida y a la WFC como la cohesión modular para pares de definición y uso común de las rebanadas dinámicas en dos o más variables de salida. Si bien las métricas de Gupta y Rao fueron desarrolladas dentro del paradigma estructural, sus principios son válidos para el paradigma orientado a objetos y en particular, para el mantenimiento de software en la esfera de ambos paradigmas.

2.4.3.2 Métricas de Mitchell y Power

Chidamber y Kemerer [Chidamber y Kemerer, 1991] definen la métrica LCOM (*Lack of Cohesion Metric* o Métrica de falta de cohesión) sobre la cual Mitchell y Power desarrollan sus métrica de

cohesión [Mitchell y Power, 2004], RL_{COM} (*Runtime Simple LCOM* o LCOM de ejecución) y RW_{LCOM} (*Runtime call-weighted LCOM* o LCOM de ejecución ponderada).

Asumiendo *n* métodos $\{M_n\}$ en una clase e *i* variables de instancia $\{I_i\}$ definimos los conjuntos:

$$P = \{(I_i, I_j) | (I_i \cap I_j) = \emptyset\}$$

$$Q = \{(I_i, I_j) | (I_i \cap I_j) \neq \emptyset\}$$

P es el conjunto de pares de métodos que no comparten variables de instancia y Q es el conjunto de pares de métodos que comparten variables de instancia. A partir de estos conjuntos se define a LCOM como:

$$LCOM = \begin{cases} |P| - |Q|, si\ |P| > |Q| \\ 0\ en\ caso\ contrario \end{cases}$$

A partir de esta métrica, definida por Chidamber y Kemerer, Mitchell y Power definen a la métrica R_{LCOM} como una extensión de LCOM que considera únicamente las variables de instancia distintas efectivamente accedidas en ejecución y a la métrica RW_{LCOM} que además considera la cantidad de veces que estas variables son utilizadas. De forma numérica:

$$P^R = \{(I_i^R, I_j^R) | (I_i^R \cap I_j^R) = \emptyset\}$$

$$Q^R = \{(I_i^R, I_j^R) | (I_i^R \cap I_j^R) \neq \emptyset\}$$

$$R_{LCOM} = \begin{cases} |P^R| - |Q^R|, si\ |P^R| > |Q^R| \\ 0\ en\ caso\ contrario \end{cases}$$

Siendo $\{I_i^R\}$ el conjunto de variables de instancias efectivamente accedidas. Y siendo N_i el número de veces que el método M_i accede a una variable de instancia I_i, además de:

$$P^W = \sum_{1 \leq i,j \leq n} \{N_i + N_j | (I_i \cap I_j = \emptyset)$$

$$Q^W = \sum_{1 \leq i,j \leq n} \{N_i + N_j | (I_i \cap I_j \neq \emptyset)$$

Mitchell y Power definen:

$$RW_{LCOM} = \begin{cases} |P^W| - |Q^W|, si\ |P^W| > |Q^W| \\ 0\ en\ caso\ contrario \end{cases}$$

Resulta interesante aclarar que si bien la métrica LCOM, perteneciente a la suite de métricas de Chidamber y Kemerer, se encuentra muy difundida y es aceptada por la comunidad de ingenieros de software, es una métrica inválida teóricamente tal como demostraron Hitz y Montazeri [Hitz y Montazeri, 1996].

2.4.3.3 Métricas de Gupta y Chhabra

Gupta y Chhabra [Gupta y Chhabra, 2011] definen una serie de métricas de cohesión que, además de ser calculadas dinámicamente a nivel de objetos, tienen en cuenta factores de la orientación a objetos como el polimorfismo y la herencia.

A los efectos de obtener datos en tiempo real, los autores desarrollaron un aplicativo para analizar dinámicamente el software en estudio. Este aplicativo fue desarrollado bajo el paradigma de programación orientada a aspectos y fue utilizado para el estudio de veinte aplicaciones Java, lo que les permitió realizar una validación empírica de sus métricas, la cual demuestra que sus métricas tienen una mayor precisión que las existentes hasta el momento [Tahir y MacDonell, 2012].

Siendo una clase c, un objeto o ($o \in O(c)$) con un número total de atributos m en donde |A(c)|=m y |A(o)|=m y un número total de métodos n en donde $|M_N(c)|=n$ y |M(o)|=n, los autores definen las siguientes métricas:

- DC_AM_x (*Dynamic Cohesion due to the Write Dependency of Attributes on Methods* o Cohesión dinámica producto de la dependencia de escritura de los atributos en métodos): este tipo de cohesión dinámica se produce cuando un método de un objeto escribe en un atributo del mismo objeto durante la ejecución de un programa. De forma numérica se define como: $DC\_AM_X(o) = \begin{cases} 0 \; si \; m = 0, n = 0 \\ \frac{\sum_{i-1}^{m} \sum_{j-1}^{n} r_W^R(e_i^R . e_j^R)}{mxn} \end{cases}$ en donde $e_i^R \in A(o) \wedge e_j^R \in M_N(o) \wedge o \in O(c)$
- DC_MA_x (*Dynamic Cohesion due to Read dependency of Methods on Attributes* o Cohesión dinámica producto de la dependencia de lectura de los métodos en atributos): esta métrica de cohesión dinámica mide la situación en la cual un método de un objeto lee un atributo del objeto en la ejecución del programa. De forma numérica: $DC\_MA_X(o) = \begin{cases} 0 \; si \; n = 0, m = 0 \\ \frac{\sum_{i-1}^{n} \sum_{j-1}^{m} r_R^R(e_i^R . e_j^R)}{nxm} \end{cases}$ en donde $e_j^R \in A(o) \wedge e_i^R \in M_N(o) \wedge o \in O(c)$
- DC_MM_x (*Dynamic Cohesion due to call dependency between methods* o Cohesión dinámica producto de la dependencia de llamadas entre métodos): este tipo de cohesión se produce en la ejecución de un programa cuando un método m_i llama a un método m_j. De forma numérica

se define como: $DC\_MM_X(o) = \begin{cases} 0\ si\ n = 0 \\ \frac{\sum_{i-1}^{n}\sum_{j-1\wedge i\neq j}^{n} r_C^R(e_i^R.e_j^R)}{nx(n-1)} \\ 1\ si\ n = 1 \end{cases}$ en donde $e_i^R \in M_N(o) \wedge e_j^R \in M_N(o) \wedge o \in O(c)$

- DC_AA_x (*Dynamic Cohesion due to reference dependency between attributes* o Cohesión dinámica producto de la dependencia de referencia entre atributos): este tipo de cohesión dinámica se produce en la ejecución de un programa cuando dos atributos son llamados dentro del mismo método. Se define como: $DC\_AA_X(o) = \begin{cases} 0\ si\ m = 0 \\ \frac{\sum_{i-1}^{m-1}\sum_{j-i-1}^{m} r_{RF}^R(e_i^R.e_j^R)}{nxmx(m-1)/2} \\ 1\ si\ m = 1 \end{cases}$ en donde $e_i^R \in A(o) \wedge e_j^R \in A(o) \wedge o \in O(c)$

2.4.3.4 Comparativa de métricas dinámicas de cohesión

Si bien la métrica de Gupta y Rao fue desarrollada para el paradigma estructural y las métricas de Mitchell y Power y Gupta y Chhabra para el paradigma orientado a objetos, todas coinciden en medir la cohesión a partir de las variables a la cual acceden los métodos. La métrica de Gupta y Rao está fuertemente orientada al mantenimiento de software ya que se basa en el concepto de rebanamiento dinámico de software, que posee diversas técnicas de aplicación. Mitchell y Power realizan un aporte muy importante a partir de la modificación de una métrica de Chidamber y Kemerer, en la cual no se orientan al mantenimiento de software sino al análisis de trazas de ejecución. Gupta y Chhabra, por su lado, proveen métricas validadas empíricamente además de proveer una aplicación desarrollada específicamente para el análisis dinámico.

2.4.4 Métricas dinámicas de complejidad

El concepto de complejidad del software engloba diversas propiedades del software y, de forma análoga al concepto de calidad de software, no tiene una definición estandarizada. Kaur *et al.* [Kaur *et al.*, 2009] lo definen como el grado en el cual un sistema o componente de un sistema tiene un diseño o implementación que es difícil de entender y verificar. Se han desarrollada diversas métricas de complejidad, cada una de las cuales se enfoca en una o más propiedades específicas del software. De forma general, a mayor cantidad de elementos en consideración, mayor será la complejidad de un software y por ende mayor el esfuerzo necesario para su desarrollo, comprensión o mantenimiento [Kaur *et al.*, 2009]. A continuación se analizan las métricas dinámicas de complejidad de software más representativas desarrolladas hasta el momento.

2.4.4.1 Métricas de Munson y Khoshgoftaar

A partir de una métrica estática de complejidad desarrollada por ellos mismos [Munson y Khoshgoftaar, 1992], Munson y Khoshgoftaar definen una métrica dinámica para medir la

complejidad [Khoshgoftaar *et al.*, 1993] en la cual ponderan el valor de la complejidad estática, o complejidad del código fuente, con la probabilidad de ejecución del componente a partir de trazas de ejecución real del sistema. Siendo p'_i la complejidad relativa de un módulo i y p la probabilidad de ejecución de dicho módulo, en un sistema de n módulos la complejidad dinámica definida por esta métrica será:

$$d_p = \sum_{i=1}^{n} p_i * p'_i$$

2.4.4.2 Métricas de Yacoub *et al.*

Al igual que con las métricas EOC e IOC, las métricas de complejidad de Yacoub *et al.* [Yacoub *et al.*, 1999] están basadas en el lenguaje de modelado ROOM (Real-Time Object Oriented Modeling) y realizan sus cálculos a partir de los modelos de simulación del mismo.

A partir de un diagrama ROOM para un objeto determinado o_j, el cual se ejecuta a partir de diferentes entradas, transiciones y salidas, se realiza un gráfico de control de flujo sobre el cual se aplica la métrica estática de complejidad ciclomática [McCabe, 1976]. Además, y a partir de estos gráficos, se obtiene una determinada probabilidad de ejecución PS en un escenario x. De forma numérica:

$$OCPX(o_i) = \sum_{x=1}^{|X|} PS_x * ocpx_x(o_i)$$

2.4.4.3 Métricas de Mathur *et al.*

Mathur *et al.* [Mathur *et al.*, 2010] definen una métrica de complejidad dinámica a la que denominan *Runtime Complexity per* object (RuCDep o complejidad por objeto en ejecución) a partir de los puntos de decisión en la ejecución de un programa. Su operación es análoga a la de la métrica de complejidad ciclomática pero analizada desde un punto de vista dinámico. La métrica consiste en realizar el siguiente cálculo para todos los objetos que son instanciados:

$$RuCDep = \frac{puntos\ de\ decisión\ de\ un\ objeto + 1}{cantidad\ de\ objetos}$$

En donde:

$$puntos\ de\ decisión\ de\ un\ objeto = \sum_{i}^{n} puntos\ \ de\ decisión\ de\ un\ método$$

En donde hay n métodos con puntos de decisión en un objeto determinado.

2.4.4.4 Comparativa de métricas dinámicas de complejidad

Las métricas de Munson y Khoshgoftaar son las únicas de las analizadas que fueron desarrolladas específicamente para el paradigma procedimental aunque su aplicación en el paradigma orientado a objetos es sencilla. Las métricas de Yacoub *et al.* y de Mathur *et al.* fueron desarrolladas para el paradigma orientado a objetos.

Por otro lado, las métricas de Yacoub *et al.* están desarrolladas para su aplicación sobre modelos de diseño mientras que las de Munson y Khoshgoftaar y Mathur *et al.* son de aplicación directa sobre trazas de ejecución del software.

2.4.5 Métricas dinámicas de polimorfismo

Se define como polimorfismo a la propiedad mediante la cual es posible enviar mensajes sintácticamente iguales a objetos distintos. El polimorfismo es una de las características del paradigma orientado a objetos que solo puede ser analizada dinámicamente. En las siguientes secciones se analizan las métricas más significativas para la medición del polimorfismo en ejecución:

2.4.5.1 Métricas de Dufour *et al.*

Dufour *et al.* [Dufour *et al.*, 2003] proponen una serie de métricas dinámicas específicamente desarrolladas para aplicaciones Java pero que son fácilmente replicables en cualquier lenguaje dentro del paradigma orientado a objetos. Estas métricas son aplicables únicamente en ejecución, ya que por ejemplo para el caso de Java hacen el análisis mediante las llamadas invokeVirtual o invokeInterface en bytecode (instrucciones de java para la máquina virtual).

Las tres primeras métricas que desarrollan miden el número de instrucciones polimórficas potenciales, pero no dicen nada de si efectivamente son llevadas a cabo o no. Las siguientes seis métricas solo consideran las ejecuciones reales de las instrucciones polimórficas, considerando tres de estas métricas para polimorfismo de recepción (*receiver polymorphism*) y las tres restantes para polimorfismo de objetivo (*target polymorphism*). A continuación se describen estas métricas:

- polymorphysm.appCallSites.value: número total de llamadas diferentes ejecutadas sin contar instrucciones de invocación estáticas. Solo se consideran las llamadas específicas de la aplicación. Dentro de Java, esta métrica es útil para determinar el tamaño de la aplicación ya que cuenta los métodos virtuales, lo que en este lenguaje aplica a todos los métodos por defecto, aún si tienen una sola implementación.

- polymorphysm.CallSites.value: es la misma métrica mencionada precedentemente, pero considerando además de las llamadas específicas de la aplicación las llamadas a las librerías del sistema.
- polymorphysm.appInvokeDensity.value: representa la densidad de llamadas polimórficas y se calcula con el número de llamadas (invokeVirtual o invokeInterface en Java) por cada mil líneas de bytecode.
- polymorphysm.appReceiverArity.bin: en forma porcentual, y partiendo de las trazas de ejecución a partir de las cuales es posible determinar las llamadas efectivas a los métodos, esta métrica indica el porcentaje de llamadas que tienen uno, dos o más de dos tipos de recepción.
- polymorphysm.appReceiverArityCalls.bin: es similar a la métrica mencionada precedentemente pero considerando todas las llamadas que se generan a partir de la llamada a un método, incluyendo aquellos con uno, dos o más de dos tipos de recepción. Esta métrica sirve para medir la importancia de las llamadas polimórficas.
- polymorphysm.appReceiverCacheMissRate.value: esta métrica representa, como porcentaje, la frecuencia con la que un método cambia de tipo de recepción. Está altamente influenciada por el orden en el cual se realizan las llamadas.
- polymorphysm.appTargetArity.bin: esta métrica representa el porcentaje de llamadas que tienen uno, dos o más de dos métodos objetivo diferentes.
- polymorphysm.appTargetArityCalls.bin: es similar a la métrica mencionada precedentemente pero contemplando todas las llamadas que se producen a partir de una llamada a un método, incluyendo métodos con uno, dos o más de dos tipos de recepción.
- polymorphysm.appTargetCacheMissRate.value: está métrica indica, en forma porcentual, la frecuencia en la que una llamada cambia entre métodos de destino. Al igual que la métrica polymorphysm.appReceiverCacheMissRate.value, también está altamente influenciada por el orden en el cual se realizan las llamadas.

2.4.5.2 Métricas de Choi y Tempero

Choi y Tempero [Choi y Tempero, 2007] desarrollan unas métricas simples para el cálculo del reuso y la reusabilidad. Para poder definir estas métricas previamente generan una métrica de polimorfismo a la que denominan índice de comportamiento polimórfico (PBI o *Polymorphic Behaviour Index*) el cual definen como:

$$PBI = \frac{P}{Mensajes\ totales}$$

En donde:

- Mensajes totales = (P + NP)
- P = mensajes polimórficos únicos distintos ejecutados
- NP = mensajes no polimórficos distintos ejecutados

Los autores diferencian los mensajes polimórficos de los no polimórficos ya que los primeros se producen cuando la interface declarada y la clase llamada son diferentes, caso contrario son mensajes no polimórficos. La interface declarada es la interface de la variable que se declara en el código fuente mientras que la clase llamada es la clase en la cual se encuentra el método llamado.

2.4.5.3 Métricas de Sandhu y Singh

Sandhu y Singh [Sandhu y Singh, 2008] desarrollan una serie de once métricas para analizar los diferentes aspectos del polimorfismo en la ejecución de un programa. Las métricas son similares a las desarrolladas por Dufour *et al.*, y al igual que éstas, fueron desarrolladas para el lenguaje Java pero son replicables en otros lenguajes orientados a objetos.

Dos métricas no contempladas inicialmente por Dufour *et al.*, son la métrica DPA (*Dynamic Polymorphism in Ancestors* o polimorfismo dinámico en predecesores) que es la suma de la cantidad de miembros de funciones dinámicas polimórficas en predecesores que aparecen en las diferentes clases y la métrica DPD (*Dynamic Polymorphism in Descendants* o polimorfismo dinámicos en sucesores) que es la suma de la cantidad de miembros de funciones dinámicas polimórficas en sucesores que aparecen en las diferentes clases.

2.4.6 Otras métricas dinámicas

Se han desarrollado métricas dinámicas sobre otros aspectos del software que no fueron analizados en las secciones anteriores. Algunas de estas métricas son: las métricas dinámicas para interfaces gráficas de usuario [Mitchell y Power, 2004], las métricas dinámicas para la valoración del riesgo en el desarrollo de software [Yacoub *et al.*, 1999], las métricas dinámicas basadas en requerimientos [Cleland-Hunang *et al.*, 2001], las métricas dinámicas para el agrupamiento de objetos [Cho *et al.*, 1998], la métrica para el cálculo del *churn* del código [Burrows *et al.* 2011] y las métricas de funcionalidad y características de aplicaciones web [Mendes *et al.*, 2005], entre otras.

2.4.7 Métricas pseudodinámicas

Con el objetivo de obtener las ventajas de las métricas dinámicas (mayor precisión) y de las métricas estáticas (mayor facilidad de cálculo y disponibilidad en etapas tempranas del desarrollo de software), Gunnalan *et al.* [Gunnalan *et al.*, 2005, Chhabra y Gupta, 2010] proponen el uso de métricas pseudodinámicas.

Las métricas pseudodinámicas consisten en el cálculo de métricas estáticas y la posterior aplicación de perfiles de operación para ajustar los valores obtenidos. Estás métricas, de propiedades análogas a las métricas dinámicas, pueden ser obtenidas en etapas tempranas del desarrollo de software.

2.5 Métricas a nivel de diseño

En esta sección se definen las métricas a nivel de diseño y se describen las principales diferencias con las métricas de otras etapas del proceso de desarrollo (sección 2.5.1). Posteriormente se enumeran y describen las principales métricas a nivel de diseño, agrupadas según la clasificación propuesta por Archer y Stinson [Archer y Stinson, 1995] y abarcando las métricas a nivel de sistema (sección 2.5.2), a nivel de acoplamiento y uso (sección 2.5.3), a nivel de herencia (sección 2.5.4), a nivel de clase (sección 2.5.5) y, finalmente, a nivel de métodos (sección 2.5.6).

2.5.1 Definición de métricas a nivel de diseño y diferencias con métricas de otras etapas en el proceso de desarrollo

En función del momento en el cual se aplican las métricas dentro del ciclo de vida de desarrollo de software las métricas pueden clasificarse en métricas para el modelo de análisis, métricas para el modelo de diseño, métricas para el código fuente y métricas para pruebas [Pressman, 2005].

Las métricas para el modelo de análisis son aquellas orientadas a la medición de los modelos de análisis e incluyen las métricas de funcionalidad entregada, que son una medida indirecta de la funcionalidad del software; las métricas de tamaño del sistema, que a partir de la información que manejará el sistema dan medidas del tamaño del mismo y las métricas de calidad de la especificación, que brindan una medida del grado de completitud de la especificación de requisitos.

Las métricas para el modelo de diseño, en cambio, tienen como objetivo cuantificar los atributos de diseño, permitiendo la toma de decisiones sobre el mismo. Pressman [Pressman, 2005; Baroni, 2002], las clasifica en métricas arquitectónicas, que son aquellas métricas que analizan la arquitectura del programa haciendo énfasis en su estructura y la eficiencia de sus componentes, en general no analizando el contenido de los componentes sino que concentrándose en las relaciones entre los mismos; las métricas a nivel de componente, que de forma opuesta a las métricas arquitectónicas, miden características internas de los mismos, como la complejidad; las métricas de diseño de la interfaz, que son aquellas métricas que estudian la usabilidad y la calidad de las interfaces con las cuales interactúa el usuario y las métricas especializadas en el diseño orientado a objetos, las cuales miden características específicas de clases y la comunicación y colaboración entre las mismas, enfocándose en características propias de este paradigma.

Las métricas para el código fuente son aquellas que además de enfocarse en aspectos inherentes del código fuente como su complejidad, consideran otras características como la facilidad de mantenimiento. Se clasifican en métricas de Halstead [Halstead, 1977], que proporcionan diferentes medidas para un aplicativo de software; métricas de complejidad, que son aquellas que miden la complejidad lógica del código fuente y las métricas de longitud o tamaño, que proporcionan medidas del tamaño del software, desde diferentes puntos de vista.

La última clasificación es la de métricas para pruebas, que son métricas que permiten guiar los procesos de prueba además de medir la eficacia de las mismas. Dentro de estas métricas encontramos a las métricas de cobertura de instrucciones y ramas que permiten medir la cobertura de las pruebas sobre el código del programa; a las métricas relacionadas con los defectos, que tienen como objetivo el cálculo de defectos de software; las métricas de efectividad de pruebas, que proporcionan medidas de la efectividad de las pruebas realizadas y las métricas de proceso que son aquellas que se determinan a medida que se aplican las pruebas.

Tal como se mencionó anteriormente, es deseable realizar mediciones del software lo antes posible en el ciclo de vida del desarrollo de software de forma tal de poder asegurar su calidad y de guiar el proceso de toma de decisiones del desarrollo. Los problemas presentes en los artefactos desarrollados en las fases iniciales de desarrollo de software se propagan a los artefactos de las siguientes fases, en donde son más costosos de identificar y corregir [Boehm, 1981]. Los paradigmas más recientes de desarrollo de software, como MDD (Model Driven Development o Desarrollo dirigido por modelos) [Atkinson et al., 2003] y MDA (Model Driven Architecture o Arquitectura dirigida por modelos) [Kleppe et al., 2003], ponen énfasis en la necesidad de modelos de calidad desde las etapas iniciales de desarrollo de software.

Existen pocas métricas en la literatura orientadas al modelo de análisis y las métricas para pruebas en general se concentran en el proceso de pruebas en lugar de enfocarse en características del software a medir [Pressman, 2005]. Las métricas a nivel de diseño y, especialmente, las métricas a nivel de código son las más utilizadas y desarrolladas para realizar mediciones de software.

Se han realizado diversos estudios comparando las métricas a nivel de diseño con las métricas a nivel de código a fin de determinar diversos factores y características del software. Si bien las métricas de diseño poseen la evidente ventaja de poder aplicarse en una etapa más temprana del proceso de desarrollo de software, las métricas a nivel de código presentan mayor precisión en sus resultados [Jiang *et al.*, 2008].

Zhao *et al.* [Zhao *et al.*, 1998] realizaron un estudio para comparar las métricas a nivel de diseño con las métricas a nivel de código en lo que respecta a la detección de errores de software llegando a la

conclusión de que las métricas a nivel de diseño tienen una performance tan buena como las métricas a nivel de código y que existe una pequeña mejora si ambas métricas son utilizadas en conjunto. En el estudio destacan el hecho de que existe una relación lógica entre los conceptos existentes en las métricas a nivel de diseño y las métricas a nivel de código.

El estudio realizado por Jiang *et al.* [Jiang *et al.*, 2008], provee curvas ROC (acrónimo de *Receiver Operating Characteristic* o Característica Operativa del Receptor), que constituyen una representación gráfica de la razón de verdaderos positivos frente a la razón de falsos positivos según la variación de un umbral de discriminación. Para el estudio se compararon el uso de métricas a nivel de diseño, de métricas a nivel de código y del uso de ambas para la detección de errores en el software. En la figura 2.4 se puede observar la curva ROC para el caso de detección de fallas de software en un módulo de mejoras de seguridad para un software de cabina de pilotos, desarrollado en C++. En la figura, PD es la probabilidad de detección y PF es la probabilidad de un falso positivo.

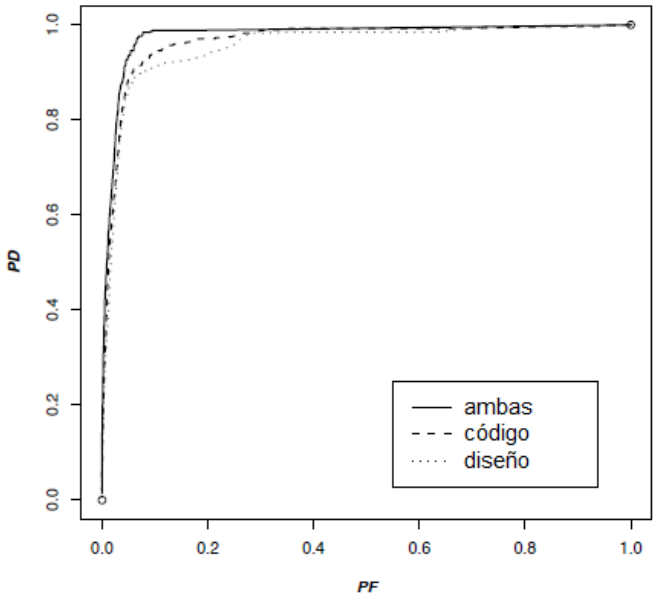


Figura 2.4 – Curva ROC para la detección de errores de software según métricas de diseño, código y ambas

En lo que respecta a las métricas dinámicas a nivel de diseño y las métricas dinámicas a nivel de código, el desarrollo de las primeras se encuentra limitado a las métricas propuestas por Yacoub *et al.* [Yacoub *et al.*, 1999] que fueron analizadas en la sección 2.4. Las métricas dinámicas a nivel de código poseen un desarrollo mucho mayor, encontrando por ejemplo métricas de acoplamiento [Mitchell y Power, 2005; Gunnalan et al., 2005; Hassoun et al., 2005; Arisholm et al., 2004; Chidamber y Kemerer, 1994]; métricas de cohesión [Gupta y Chhabra, 2011; Safari-Sharifadabi y Constantinides, 2008; Gupta y Rao, 2001; Cho et al.¸1998; Chidamber y Kemerer, 1994] y métricas de complejidad estructural [Ma et al., 2005; Yacoub y Ammar, 2002; Yacoub et al., 1999; Chidamber y Kemerer, 1994; Munson y Khoshgoftaar, 1992], entre otras.

Rodriguez y Harrison [Rodriguez y Harrison, 2001], con el objetivo de abarcar las características y la granularidad del paradigma orientado a objetos, utilizan una taxonomía diferente, descripta originalmente por Archer y Stinson [Archer y Stinson, 1995] la cual clasifica a las métricas de diseño en métricas a nivel de sistema, métricas a nivel de acoplamiento y uso, métricas a nivel de herencia, métricas a nivel de clase y métricas a nivel de método. Esta taxonomía está desarrollada a partir de las características y propiedades del paradigma orientado a objeto, capturándolas de forma jerárquica.

A continuación se describen las métricas a nivel de diseño más significativas, dentro del paradigma orientado a objetos, tomando como base la clasificación propuesta por Archer y Stinson [Pressman, 2005; Rodriguez y Harrison, 2001; Archer y Stinson, 1995].

2.5.2 Métricas a nivel de sistema

El primer nivel de la clasificación abarca al sistema y sus componentes como un conjunto. Las métricas a nivel de sistema son aquellas relacionadas, por ejemplo, con el comportamiento externo de las clases y la relación entre ellas. A continuación se analizan la suite de métricas MOOD (*Metrics for Object Oriented Design* o Métricas para el diseño orientado a objetos) propuestas por Abreu y Melo [Abreu y Melo, 1996], las cuales operan a nivel de sistema. Las métricas MOOD fueron validadas teórica y empíricamente [Genero *et al.*, 2005].

2.5.2.1 MHF

La métrica MHF (*Method Hiding Factor* o Factor de ocultamiento de métodos) es definida como la razón existente entre la suma de invisibilidades de los métodos en todas las clases sobre la cantidad de métodos totales del sistema. La invisibilidad de un método, a su vez, se define como el porcentaje de clases desde las cuales el método no es accesible.

Esta métrica es útil para medir el encapsulamiento del sistema y no considera métodos heredados en los cálculos. Abreu y Melo [Abreu y Melo, 1996] encontraron que a mayor MHF se espera una menor densidad de defectos y un menor costo de reparación de los mismos.

De forma numérica:

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}$$

En donde M_d es el número de métodos definidos en una clase, $V(M_{mi}) = \frac{\sum_{j=1}^{TC} visible(M_{mi}, C_i)}{TC-1}$ y la función visible se define como 0 si el método puede ser usado por una clase o 1 en caso contrario. TC representa el número total de clases del sistema.

2.5.2.2 AHF

La métrica AHF (*Attribute Hiding Factor* o Factor de ocultamiento de atributos) es, de forma análoga a la métrica MHF, la razón entre la suma de invisibilidades de los atributos de todas las clases sobre la cantidad de atributos totales del sistema. Es otra medida del encapsulamiento y de forma *purista* debería ser siempre el 100% ya que forma general se espera que los objetos no hagan pública su implementación [Rodriguez y Harrison, 2001].

De forma numérica:

$$AHF = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{A_d(C_i)}(1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}$$

En donde A_d es el número de atributos de una clase, $V(A_{mi}) = \frac{\sum_{j=1}^{TC} visible(A_{mi}, C_i)}{TC-1}$ y la función visible se define como 0 si el atributo puede ser accedido por una clase o 1 en caso contrario. TC representa el número total de clases del sistema.

2.5.2.3 MIF

MIF (*Method Inheritance Factor* o Factor de herencia de métodos) se define como la razón o proporción entre la cantidad total de métodos heredados en todas las clases del sistema y la cantidad total de métodos (locales y heredados) de todas las clases del sistema. Es una medida de la herencia y la reusabilidad en el sistema.

De forma numérica se define como:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

En donde $M_a(C_i) = M_d(C_i) + M_i(C_i)$ y $M_d(C_i)$ representa todos los métodos declarados en una clase, $Ma(C_i)$ los métodos que pueden ser invocados desde una clase, $M_i(C_i)$ los métodos heredados por una clase y que no fueron sobrescritos y TC es el número total de clases de un sistema.

2.5.2.4 AIF

De forma análoga a MIF, AIF (*Attribute Inheritance Factor* o Factor de herencia de atributos) es la proporción existente entre la cantidad de atributos heredados en todas las clases del sistema y la cantidad total de atributos (locales y heredados) de todas las clases del sistema. Es una medida de la herencia y la reusabilidad.

De forma numérica:

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

En donde $A_a(C_i) = A_d(C_i) + A_i(C_i)$ y $A_d(C_i)$ representa todos los atributos declarados en una clase, $Aa(C_i)$ los atributos que pueden ser invocados desde una clase, $A_i(C_i)$ los atributos heredados por una clase y que no fueron sobrescritos y TC es el número total de clases de un sistema.

2.5.2.5 PF

El PF (*Polymorphism Factor* o Factor de Polimorfismo) es una medida del polimorfismo potencial y se define como la razón entre el número actual de situaciones polimórficas posibles en una clase y el número total de situaciones polimórficas de dicha clase. PF es el número de métodos que redefinen a métodos heredados, dividido por el máximo número posible de situaciones polimórficas.

Esta métrica es también una medida indirecta del uso del sistema de tipos dinámicos. En sistemas sin herencia esta métrica tiene un valor no definido ya que el denominador de la razón se hace cero.

De forma numérica:

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i).DC(C_i)]}$$

En donde: $M_d(C_i) = M_n(C_i) + M_o(C_i)$ y $M_n(C_i)$ es el número de nuevos métodos, $M_o(C_i)$ es el número de métodos sobrescritos, $DC(C_i)$ es la cantidad de descendientes y TC el número total de clases del sistema.

2.5.2.6 CF

La métrica CF (*Coupling Factor* o Factor de Acoplamiento) se define como la razón entre la cantidad total de acoplamientos del sistema y la cantidad de acoplamientos del sistema no relacionados con la herencia. El acoplamiento es normalmente visto como un aumentador de la complejidad, reduciendo el encapsulamiento y la reusabilidad, además de a la vez limitar el entendimiento del software y su facilidad de mantenimiento [Rodriguez y Harrison, 2001], es por esto que es importante conocer el número de comunicaciones entre clases, representado por esta métrica.

De forma numérica se calcula como:

$$CF = \frac{\sum_{i=1}^{TC} \left| \sum_{j=1}^{TC} cliente(C_i, C_j) \right|}{TC^2 - TC}$$

En donde la función cliente se define como 1 si la clase C_i accede a la clase C_j a través de al menos un método no heredado (además de que $C_i \neq C_j$) y 0 en caso contrario. Al igual que en las demás métricas de MOOD, TC es el número total de clases del sistema.

2.5.3 Métricas a nivel de acoplamiento y uso

Definido el acoplamiento como el uso de métodos o atributos de una clase por parte de otra, es posible afirmar que la interacción entre clases normalmente da origen a la existencia de subsistemas dentro de un sistema. Es deseable conocer las características de este subsistema y la interacción presente en el mismo ya que pueden complicar el diseño de un programa software [Rodriguez y Harrison, 2001].

La métrica más representativa de este nivel es la métrica CBO (*Coupling Between* Objects o Acoplamiento a nivel de objetos) definida en la suite de métricas de Chidamber y Kemerer [Chidamber y Kemerer, 1994].

2.5.3.1 CBO

Esta métrica, definida a nivel de una clase, representa la cantidad de clases a la cual esta se encuentra acoplada, es decir, que tiene una relación de uso a través de métodos no heredados con otras clases de las cuales depende.

Chidamber y Kemerer sugieren que esta métrica representa el esfuerzo necesario para el mantenimiento y el desarrollo de pruebas del sistema. A mayor acoplamiento es mayor la complejidad del sistema y más difícil de aplicar reusabilidad. Las clases deberían ser independientes de forma tal de promover el reuso y el encapsulamiento.

2.5.4 Métricas a nivel de herencia

Las clases de un sistema son representadas normalmente en un diagrama de clases en el cual se pueden observar distintas características del diseño del sistema como la herencia. El enrejado de clases, junto a su profundidad y ancho, presente en un diagrama de clases permite determinar características del diseño del software que deben ser medidas.

La herencia es una de las características más interesantes del paradigma orientado a objetos pero que, en general, tiene un impacto negativo en el diseño de sistemas ya que dificulta el entendimiento y hace difícil el mantenimiento del software, debido a que las interfaces de las clases que heredan son más difíciles de modificar.

Las métricas DIT (*Depth of Inheritance Tree* o Profundidad del árbol de herencia) y NOC (*Number of Children* o Número de hijos) de Chidamber y Kemerer [Chidamber y Kemerer, 1994], la métrica SIX (*Specialisation Index per Class* o Índice de especialización por clase) de Lorenz y Kidd [Lorenz

y Kidd, 1994] y la métrica ANA de Bansiya y Davis [Bansiya y Davis, 2002] son las más representativas de este nivel.

2.5.4.1 DIT

Esta métrica mide el nivel máximo o la profundidad de la herencia en la jerarquía de una clase. En el diagrama de clases las clases que no heredan de ninguna se consideran al nivel cero y a partir de ellas cada herencia adiciona uno al nivel o métrica.

Esta métrica permite medir de una forma muy sencilla la complejidad de una clase ya que se asume que a un mayor nivel, mayores métodos y atributos tendrá la misma.

2.5.4.2 NOC

Esta métrica mida la cantidad de subclases que heredan directamente de una superclase particular. Es una forma de medir la reusabilidad de una clase particular y de su importancia general en el sistema.

2.5.4.3 SIX

Esta métrica mide el grado en el cual una clase sobrescribe funcionalidad de su superclase, es decir, de la clase a partir de la cual hereda. Si bien la métrica está desarrollada para lenguajes que acepten herencia de varias clases, la mayoría de los lenguajes desarrollados hasta el momento no implementa esta característica.

De forma numérica se calcula como:

$$SIX = \frac{Métodos\ sobrescritos * DIT}{Número\ total\ de\ métodos}$$

El producto que se realiza en el numerador da mayor peso a los métodos sobrescritos que se dan en niveles más bajos del diagrama de clase ya que las clases más bajas deberían estar más especializadas y minimizar el reemplazo del comportamiento de base, es decir, de niveles más altos en el diagrama de clases.

Esta métrica es una forma de medir la calidad de la herencia y puede servir para detectar casos puntuales de exceso de sobrescritura. Se espera que, de una forma general, una clase agregue métodos a su superclase en lugar de que los reescriba.

2.5.4.4 ANA

ANA (*Average Number of Ancestors* o Número promedio de ancestros) es una forma de medir la abstracción en un sistema. Se calcula a partir del cálculo promedio de los ancestros de todas las clases.

2.5.5 Métricas a nivel de clase

El estudio, y por ende la medición, de las clases en el diseño de sistemas es fundamental. Las clases contienen a los métodos y atributos y, además de su funcionamiento interno, tienen relación con otras clases. El funcionamiento interno de una clase, es decir, la relación entre sus métodos y atributos es de suma importancia ya que la estructura de conectividad de un sistema es más importante que el contenido de los métodos individuales [Churcher y Shepperd, 1995].

Las métricas más representativas de este nivel pertenecen a la suite de métricas desarrollada por Chidamber y Kemerer [Chidamber y Kemerer, 1994], a las métricas definidas por Li y Henry [Li y Henry, 1993] y a las métricas definidas por Lorenz y Kidd [Lorenz y Kidd, 1994].

2.5.5.1 RFC

La métrica RFC (*Response for a Class* o Respuesta de una clase), desarrollada por Chidamber y Kemerer, se define como la cantidad de métodos que pueden ser invocados en respuesta a un mensaje hacia un objeto de la clase o por algún método en la clase, incluyendo a todos los métodos accesibles desde la jerarquía de clase. RFC cuenta la cantidad de llamadas a otras clases a partir de una en particular.

De acuerdo a Chidamber y Kemerer esta métrica es un buen indicador de la complejidad de una clase determinada a través de sus métodos y de la comunicación con otras clases.

2.5.5.2 WMC

La métrica WMC es definida por Chidamber y Kemerer como la complejidad individual de una clase. En su desarrollo de métricas no dan ninguna definición o métrica para el cálculo de la complejidad por lo que usada críticamente solo sería una métrica del tamaño de una clase [Harrison *et al.*, 1997].

Siendo una clase C_1 con M_n métodos de respectivas c_n complejidades, WMC se calcula como:

$$WMC\ (C_1) = \sum_{i=1}^{n} c_i$$

2.5.5.3 LCOM

Esta métrica, desarrollada por Chidamber y Kemerer, ya fue explicada en la sección 2.4.2.2 dado que es la base para el desarrollo de las métricas de Mitchell y Power. Desde el punto de vista de diseño, esta métrica es una medida de la calidad de la cohesión de una clase debido a que permite medir el número de atributos comunes usados por diferentes métodos. Es también una medida de la abstracción de una clase.

Henderson-Sellers [Henderson-Sellers, 1995] propone una nueva forma de medir LCOM partiendo de dos críticas a esta métrica. En primer lugar la métrica da lugar a dos clases con LCOM = 0 aún una teniendo más variables que otra y en segundo lugar el hecho de que Chidamber y Kemerer no proporcionan una guía para la interpretación de sus valores.

La métrica propuesta por Henderson-Sellers se calcula de la siguiente forma:

$$LCOM' = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1 - m}$$

En donde hay M_m métodos con A_a atributos y el número de métodos que accede a un atributo A_j es $\mu(A_j)$. Si todos los métodos acceden a todos los atributos, LCOM' se hace cero ya que $\sum \mu(A_j) = ma$, indicando una perfecta cohesión. Si, en cambio, cada método accede a solo un atributo, LCOM' se hace uno ya que $\sum \mu(A_j) = a$, indicando una falta de cohesión.

2.5.5.4 DAC

Esta métrica, definida por Li y Henry, cuenta el número de atributos de una clase que poseen como tipo a otra clase. Es una forma de medir el acoplamiento entre clases y tiene impacto en la complejidad del sistema.

2.5.5.5 DAC'

De forma similar a la métrica anterior, esta métrica desarrollada por Li y Henry mide el número de clases que son usadas como tipos de atributos en otras clases. Permite medir el acoplamiento y la complejidad de un sistema, además de la reusabilidad.

2.5.5.6 NOM

La métrica NOM (*Number of local Methods* o Número de métodos locales), definida por Li y Henry, mide la cantidad de métodos locales de una clase. Sirve para medir el tamaño de una clase. El acrónimo de esta métrica es el mismo utilizado para la métrica *Number of Messages*, definida por Lorenz y Kidd y explicada en la sección 2.5.5.2.

2.5.5.7 SIZE2

Esta métrica, definida por Li y Henry, es la cantidad de atributos sumada a la cantidad de métodos locales (NOM) de una clase. Es una métrica que mide el tamaño y, en menor medida, la complejidad de un sistema.

2.5.5.8 APPM

La métrica APPM (*Average Parameters Per Method* o Parámetros promedio por método) fue desarrollada por Lorenz y Kidd y se define como el promedio de parámetros por método a nivel de clase. Numéricamente se define como: $APPM = Parámetros\ totales/Métodos\ totales$

2.5.6 Métricas a nivel de métodos

Los métodos y los atributos forman parte del nivel más bajo en la escala propuesta por Archer y Stinson. En general los métodos dentro del paradigma orientado a objetos están desarrollados de una forma similar a la que son desarrollados en el paradigma estructurado con la particularidad de que en el paradigma orientado a objeto los métodos son los responsables de invocar a otros objetos.

La mayoría de las métricas existentes en el paradigma estructurado de desarrollo de software son aplicables a los métodos del paradigma orientado a objetos. Cabe señalar que algunas métricas de complejidad (como la métrica de complejidad ciclomática) no son usadas extensivamente, ya que la complejidad en el paradigma orientado a objeto suele estar dada por la interacción entre métodos más que por el contenido de un método. Las métricas más significativas de este nivel fueron desarrolladas por Lorenz y Kidd [Lorenz y Kidd, 1994] y por Bansiya y Davis [Bansiya y Davis, 2002].

2.5.6.1 LOC

La métrica LOC (*Lines of Code* o líneas de código) es una de las métricas más habituales en la literatura de la materia. Desarrollada en 1994 por Lorenz y Kidd, es aplicable de diversas maneras como por ejemplo contando las líneas reales de código o el número de instrucciones y es de muy fácil aplicación. Más allá de los problemas inherentes a la métrica, la mayoría de los autores coincide en que no es una métrica recomendable para el paradigma orientado a objetos [Rodriguez y Harrison, 2001].

2.5.6.2 NOM

NOM (*Number of Messages* o número de mensajes), definida por Lorenz y Kidd, mide la cantidad de mensajes o llamadas realizadas dentro de un método, clasificadas por el tipo de mensaje. Los tipos de mensajes varían según la cantidad de parámetros existentes en la llamada, pudiendo ser unarios, binarios, etc. No debe ser confundida con la métrica definida por Li y Henry y que fue descripta en la sección 2.5.4.4.

La métrica, al igual que LOC, permite de una forma muy sencilla obtener un valor que determine de una forma primitiva la complejidad de un método.

2.5.6.3 CAMC

CAMC (*Cohesion Among Methods of Class* o Cohesión entre métodos de una clase) es una métrica desarrollada por Bansiya y Davis que se calcula a partir de la suma de la intersección de parámetros de un método con el conjunto máximo independiente de tipos de parámetros en una clase. Es una forma de medir la relación entre los distintos métodos.

2.6 Métricas para UML

En la sección 2.2 se analizaron los diagramas existentes en UML para los flujos de análisis y diseño de sistemas. La importancia de este lenguaje de modelado ha llevado a diversos autores a adaptar sus métricas para su incorporación al mismo o incluso al desarrollo de métricas específicas para UML [Genero et al., 2005]. En esta sección se analizan las métricas más significativas desarrolladas específicamente para UML.

2.6.1 Métricas de Kim y Boldyreff

Kim y Boldyreff [Kim y Boldyreff, 2002] desarrollaron una suite de veintisiete métricas para la medición de diversos factores de un software modelado en UML. Además de estas métricas, generaron una herramienta CASE (Computer Aided Software Engineering o Ingeniería de Software Asistida por computadora) llamada UMP (UML Metrics Producer o Productor de métricas UML) la cual está implementada mediante el lenguaje BasicScript para Rational Rose. Las métricas de Kim y Boldyreff son calculadas a partir de un metamodelo de UML que posee tres entidades: atributos, clases y operaciones. Las métricas, además, están divididas en cuatro categorías, las cuales se ven a continuación.

2.6.1.1 Métricas para modelos

Las métricas para modelos se utilizan para la estimación del tamaño o la cantidad de información presente en un modelo. Son útiles para medir características de un sistema que aún no se encuentra desarrollado. Las métricas para modelos son:

- NPM (Number of packages in a model o Número de paquetes en un modelo): es la cantidad de paquetes existentes en un modelo. Los paquetes son formas de agrupar elementos comunes en un modelo.
- NCM (Number of clases in a model o Número de clases en un modelo): utilizada para determinar el tamaño de un sistema se calcula contando la clases de un sistema.
- NAM (Number of Actors in a model o Número de actores en un modelo): esta métrica determina el número de actores en un modelo. Los actores son instancias de las clases en el metamodelo de UML.

- NUM (Number of Use Cases in a model o Número de casos de uso en u modelo): es otra forma de medir el tamaño de un sistema pero apuntando a la funcionalidad, determinada por la cantidad de casos de uso en u sistema.
- NOM (Number of Objects in a model o Número de objetos en un modelo): de forma análoga a la métrica NCM, mide la cantidad de objetos en un sistema. Los objetos son instancias de las clases.
- NMN (Number of Messages in a model o Número de mensajes en un modelo): los mensajes son intercambiados entre los objetos con distintos fines. La cantidad de mensajes en un sistema determina esta métrica.
- NASM (Number of associations in a model o Número de asociaciones en un modelo): es la cantidad de conexiones o vínculos entre clases.
- NAGM (Number of aggregations in a model o Número de agregaciones en un modelo): la agregación es un tipo de asociación en particular en la cual una clase forma parte de otra.
- NIM (Number of inheritance relations in a model o Número de relaciones de herencia en un modelo): es el número de relaciones de generalización existentes entre clases en el modelo. La herencia forma relaciones de generalización o especialización, dependiente del sentido desde el cual se analiza.

2.6.1.2 Métricas para clases

Las métricas para clases son aquellas que se enfocan en características de las clases como atributos, relaciones entre clases e instanciación de objetos. A continuación se detallan las mismas:

- NATC1 (*Number of attributes in a class – unweighted* o Número no ponderado de atributos en una clase): esta métrica cuenta la cantidad total de atributos en una clase.
- NATC2 (*Number of attributes in a class – weighted* o Número ponderado de atributos en una clase): esta métrica es una versión ponderada de la anterior. Cuenta los atributos de una clase pero asignando un peso específico a cada uno en función de su visibilidad. Permite dar una medida del encapsulamiento de una clase. Los autores sugieren dan como ejemplo los valores de 1 para atributos públicos, 0,5 para atributos protegidos y 0 para atributos privados.
- NOPC1 (*Number of operations in a class – unweighted* o Número no ponderado de operaciones en una clase): esta métrica cuenta los métodos totales de una clase
- NOPC2 (*Number of operations in a class – weighted* o Número ponderado de operaciones en una clase): esta métrica es una versión ponderada de la anterior. Contabiliza los métodos de una clase pero asignando un peso específico a cada uno en función de su visibilidad. Los autores sugieren la misma valuación de pesos que en la métrica NATC2.

- NASC (*Number of associations linked to a class* o Número de asociaciones vinculadas a una clase): número de asociaciones de una clase, incluyendo asociación. Es útil para estimar las relaciones estáticas entre clases.
- CBC (*Coupling between classes* o Acoplamiento entre clases): esta métrica cuenta la cantidad de asociaciones en una clase y los atributos cuyos parámetros son clases de otro tipo.
- DIT (*Depth of Inheritance* Tree o Profundidad del árbol de herencia): Los autores la definen de la misma forma que la métrica de igual nombre en la suite de métricas de Chidamber y Kemerer, vista en la sección 2.5.3.1.
- NSUPC (*Number of superclasses of a class* o Número de superclases de una clase): cuenta la cantidad de clases padre desde la cual una clase hereda. En lenguajes que permiten una sola herencia (como Java o SmallTalk) este métrica solo puede ser cero o uno.
- NSUPC* (*Number of elements in the transitive closure of the superclasses of a class* o Número de elementos en el cierre transitivo de las superclases de una clase): esta clase cuenta la cantidad de clases que se encuentran por encima de ella en el árbol de herencia. Es una forma de determinar la cantidad de clases que, en caso de ser modificadas, tendrían impacto en la clase en estudio.
- NSUBC (*Number of subclasses of a class* o Número de subclases de una clase): cuenta la cantidad de clases hijas de una clase específica.
- NSUBC* (*Number of elements in the transitive closure of the subclasses of a class* o Número de elementos en el cierre transitivo de las subclases de una clase): es la cantidad de clases que dependen en línea directa de la clase en análisis
- NMSC (*Number of messages sent by the instantiated objects of a class* o Número de mensajes enviados por los objetos instanciados de una clase): esta métrica, utilizada para determinar las clases involucradas en las interacciones en un sistema, cuenta la cantidad de mensajes enviados por los objetos instanciados de una clase.
- NMRC (*Number of messages received by the instantiated objects of a class*): esta métrica es similar a la métrica RFC desarrollada por Chidamber y Kemerer, vista en la sección 2.5.4.1

2.6.1.3 Métricas para mensajes

Los mensajes en UML representan interacciones entre objetos. Las métricas desarrolladas en esta categoría permiten medir el grado de interacciones.

- NDM (*Number of directly dispatched messages of a message* o Número de mensajes directos enviados por un mensaje): en UML un mensaje puede generar el envío de otros mensajes. Esta métrica cuenta la cantidad de mensajes que son activados a partir de un mensaje específico.

- NDM* (*Number of the elements in the transitive closure of the directly dispatched messages of a message* o Número de elementos en el cierre transitivo de los mensajes directos enviados por un mensaje): son los mensajes generados a partir de la activación de un mensaje hasta la recepción de una respuesta por parte del objeto que lo generó.

2.6.1.4 Métricas para casos de uso

Los casos representan una serie de pasos o actividades que deberán seguirse para llevar a cabo algún proceso. Son, de alguna forma, un contrato entre los *stakeholders* (actores principales del sistema) y el sistema con respecto a su comportamiento. Las métricas desarrolladas por Kim y Boldyreff para casos de uso son las siguientes:

- NAU (*Number of Actors associated to a use case* o Número de actores asociados a un caso de uso): representa la cantidad de actores que están asociados a un caso de uso y, según los autores, es una forma de determinar la importancia de un caso de uso ya que una mayor cantidad de actores indicará una mayor importancia en el sistema. Para esta métrica no se consideran las clases de sistema.
- NMU *Number of Messages associated to a use case* o Número de mensajes asociados a un caso de uso): esta métrica cuenta la cantidad de mensajes existentes dentro de un caso de uso considerando todos los escenarios posibles. Se calcula a partir de los diagramas de secuencia o los diagramas de colaboración, los cuales son isomorfos entre sí.
- NSCU (*Number of system classes associated with a use case* o Número de clases de sistema asociadas a un caso de uso): esta métrica cuenta la cantidad d clases cuyos objetos participan en un escenario de un caso de uso, sin contar los actores ya que los mismos son considerados en la métrica NAU.

2.6.2 Métricas de Marchesi

Marchesi [Marchesi, 1998] desarrolla una serie de métricas que clasifica en tres grupos: métricas de clase, métricas de paquete y métricas de sistema. Estas métricas normalmente son aplicables en la etapa de análisis y solo tienen por nombre un acrónimo sin significado definido. Dentro de las métricas de Marchesi se define a la responsabilidad de una clase como la información que mantiene o como los cálculos que realiza.

El objetivo de las métricas es medir la complejidad del sistema, evaluar el balance de responsabilidades entre clases y paquetes, y la cohesión y el acoplamiento de las entidades del sistema. Las métricas no fueron validadas teóricamente pero el autor las validó empíricamente aplicándolas a tres sistemas reales.

2.6.2.1 Métricas de clase

- CL1: es el número ponderado de responsabilidades de una clase, heredadas o no. Se calcula como $CL1 = NC_i + K_a NA_i + K_r \sum_{h \epsilon b(i)} NC_h$. En donde NC_i es el número de responsabilidades concretas de la clase C_i, NA_i es el número de responsabilidades abstractas de la dicha clase y b(i) es un arreglo cuyos elementos son los índices de las superclases de C_i.
- CL2: es el número ponderado de dependencias de una clase. Se diferencian las dependencias específicas de la clase de aquellas heredadas. Se calcula como: $CL2 = \sum_{k=1}^{N_c} {d_{jk}}^{kd} + K_e \sum_{j \epsilon b(i)}^{N_c} {d_{jk}}^{kd}$ en donde $K_d < 1$ y N_c es el número total de clases. d_{jk} es el elemento de una matriz $[D]_{NCxNC}$ que representa las dependencias de una clase C_i (cliente) con respecto a una clase C_k (servidor).

2.6.2.1 Métricas de paquete

- PK1: esta métrica define el número de dependencias entre clases pertenecientes a un paquete específico P_k, y clases pertenecientes a otros paquetes. En otras palabras, son todas aquellas clases cliente del paquete P_k que tienen clases servidoras en otros paquetes. Se define como: $PK1 = \sum_{i/p_{ik}=1} \left(\sum_{i/p_{ik} \neq 1} d_{ih} \right)$ donde $[P]_{NCxNP}$ es la matriz de clase-paquete en donde el elemento p_{ik} es 1 si la clase C_i pertenece al paquete P_k. Los demás elementos de la fila serán siempre cero.
- PK2: es opuesta a PK1 ya que considera las dependencias de las clases servidoras. Se define como: $PK1 = \sum_{i/p_{ik} \neq 1} \left(\sum_{i/p_{ik}=1} d_{ih} \right)$ con los mismos elementos de la métrica PK1 pero siendo la clase C_i una servidora.
- PK3: es el promedio de la métrica PK1 y se define como $PK3 = \frac{\sum_{k=}^{Np} (PK1)}{N_p}$ en donde N_p es el número total de paquete. Esta métrica es una aproximación al acoplamiento entre paquetes.

2.6.2.1 Métricas de sistema

- OA1: es la cantidad de clases del sistema
- OA2: es la cantidad de jerarquías de herencia en el sistema
- OA3: es el promedio ponderado de responsabilidades de clases. $OA3 = \left(\sum_{i=1}^{N_c} CL1 \right) / N_c$. Para una clase cualquiera es demostrable que OA3 ≤ CL1
- OA4: es la desviación estándar del número de responsabilidades de clases. Se define como: $OA4 = \frac{1}{N_c} \sum_{i=1}^{N_c} (CL1 - < CL1 >)^2$
- OA5: es el promedio ponderado de dependencias de clases. $OA5 = \left(\sum_{i=1}^{N_c} CL2 \right) / N_c$. Para una clase cualquiera es demostrable que OA5 ≤ CL2

- OA6: es la desviación estándar de las dependencias de clases. Se define como: $OA6 = \frac{1}{N_c}\sum_{i=1}^{N_c}(CL2 - < CL2 >)^2$
- OA7: es el porcentaje de responsabilidades heredadas con respecto al total. No se consideran las responsabilidades heredadas que son sobrescritas por la subclase. Siendo AR_k el número de responsabilidades heredadas de una clase (sin ser sobrescritas) y XRk el número total de responsabilidades de la clase (heredadas o no), definimos a OA7 como: $OA7 = \frac{\sum_{k=1}^{N_c} AR_k}{\sum_{k=1}^{N_c} XR_k}$

2.6.3 Métricas de Genero

Las métricas de Genero [Genero, 2002] se clasifican en dos grupos: las métricas de clase y las métricas de diagrama de clases. Estas métricas fueron definidas con el objetivo de medir la complejidad en un diagrama de clases considerando diferentes tipos de relaciones como asociaciones, generalizaciones, agregaciones y dependencias. Las métricas fueron validadas tanto teórica como empíricamente y hasta se ha desarrollado una utilidad para su cálculo automático a partir de diagramas de clases de Rational Rose. A continuación se describen las métricas desarrolladas por la autora:

2.6.3.1 Métricas de diagrama de clase

- NAssoc (*Number of associations metric* o Métrica de número de asociaciones) es una métrica que determina la cantidad de asociaciones en un diagrama de clases. Esta métrica permite medir la complejidad del sistema.
- NAgg (*Number of aggregations metric* o Métrica de número de agregaciones) es el número total de relaciones de agregación presentes en el diagrama. Esta métrica permite medir la complejidad del sistema.
- NDep (*Number of dependencies metric* o Métrica de número de dependencias) es el número total de relaciones de dependencia presentes en el diagrama. Esta métrica permite medir la complejidad del sistema.
- NGen (*Number of generalizations metric* o Métrica de número de generalizaciones) es el número de relaciones de generalización presentes en el diagrama. Esta métrica permite medir la complejidad del sistema.
- NGenH (*Number of generalization hierarchies metric* o Métrica de número de jerarquías de generalización): es el número de jerarquías de generalización presentes en el diagrama. Es una forma de medir el tamaño de un sistema.
- NAggH (*Number of aggregation hierarchies metric* o Métrica de número de jerarquías de agregación): es el número de jerarquías de agregación presentes en el diagrama. Es una forma de medir el tamaño de un sistema.

- MaxDIT (*Maximum DIT* o DIT máximo): es el valor máximo de la métrica DIT vista en la sección 2.5.3.1 para todo el diagrama de clases. Permite medir la longitud de un sistema.
- MaxHAgg (*Maximum Hagg* o Hagg máximo): está métrica está definida como el máximo valor HAgg para cada clase del diagrama. El valor HAgg es, para las jerarquías de agregación, el camino más largo desde una clase hasta una hoja. Permite medir la longitud de un sistema.

2.6.3.2 Métricas de clase

- NAssocC (*Number of associations per class metric* o Métrica de número de asociaciones por clase): es el número total de asociaciones que una clase tiene con otras o consigo misma. Es una forma de medir el acoplamiento.
- HAgg (*Height of a class within an aggregation metric* o Métrica del peso de una clase dentro de una agregación): es la cantidad de clases en el camino de la jerarquía de agregación desde la clase hasta una hoja. Es una forma de medir la longitud de un sistema.
- NDP (*Number of direct parts metric* o Métrica de número de partes directas): es el número total de clases que componen, de forma directa, una clase compuesta en una jerarquía de agregación. Es una forma de medir el tamaño de un sistema.
- NP (*Number of parts metric* o Métrica de número de partes): es la cantidad de partes, directas o no, que componen una clase compuesta en una jerarquía de agregación. Es una forma de medir el tamaño de un sistema.
- NW (*Number of whole metric* o Métrica de clases completas): es la cantidad de clases completas que forman parte de una clase parcial en una jerarquía de agregación. Es una forma de medir el tamaño de un sistema.
- MAgg (*Multiple aggregation metric* o Métrica de agregación múltiple): se define como el número de clases completas directas de las cuales una clase forma parte en una jerarquía de agregación.
- NDepIn (*Number of dependencies In metric* o Métrica de número de dependencias de entrada): es el número de clases que dependen de una clase dada. Es una forma de medir el acoplamiento.
- NDepOut (*Number of dependencies Out metric* o Métrica de número de dependencias de salida): es el número de clases de las cuales depende una clase dada. Es una forma de medir el acoplamiento.

2.7 Redes complejas

En esta sección se realiza una introducción a los conceptos de redes complejas (sección 2.7.1), se detallan los más recientes avances en el uso de redes complejas para el modelado de sistemas software

(sección 2.7.2) y finalmente se describen las métricas de software implementadas mediante el uso de redes complejas (sección 2.7.3).

2.7.1 Introducción a las redes complejas

Se define como una red compleja a un grafo con propiedades topológicas no triviales. Tradicionalmente el estudio de las redes complejas se mantuvo asociado al estudio de los grafos regulares hasta que a finales de la década de 1950, Erdős y Rényi [Erdős y Rényi, 1966; Erdős y Rényi, 1961; Erdős y Rényi, 1959] propusieron su modelo de grafos aleatorios, modelo que fue aceptado como el más simple y representativo de los mismos [Albert y Barabási, 2002].

De acuerdo al modelo propuesto por Erdős y Rényi, un grafo aleatorio de N nodos tendrá arcos entre dichos nodos de forma aleatoria y con una probabilidad aproximada de pN(N-1)/2. El número arcos o aristas de cada nodo (el grado del mismo) seguirá por ende una distribución normal [Wen *et al.*, 2007]. Este modelo se mantuvo vigente hasta el aumento en la capacidad de cómputo y el desarrollo de grandes bases de datos, avances que llevaron al desarrollo de nuevos conceptos [Albert y Barabási, 2002].

El primero de estos conceptos es el de *small-world* o mundo pequeño, que describe que a pesar del gran tamaño de las redes complejas, en la mayoría de redes existe un camino relativamente corto entre dos nodos cualesquiera. La distancia entre dos nodos se define como el número mínimo de arcos o aristas que conectan a ambos. Este concepto fue popularizado gracias al psicólogo Stanley Milgram [Milgram, 1967] que concluyó que entre dos personas cualesquiera de los EEUU hay una cadena de conocidos que no tiene más de seis personas en promedio. Esta propiedad, presente en la mayoría de redes complejas, no necesariamente es indicativa de algún principio de ordenamiento ya que se ha demostrado que existe hasta en grafos aleatorios, tendiendo en éstos el camino promedio a ser el logaritmo de la cantidad de nodos del grafo [Erdős y Rényi, 1966].

El segundo concepto es el de *clustering* o agrupamiento, cuantificado a través del coeficiente de clustering definido por Watts y Strogatz [Watts y Strogatz, 1998]. El concepto surge del concepto de cliqué en grafos no dirigidos, el cual define a un conjunto de vértices tal que para todo par de vértices existe una arista que los conecta. Dentro de las redes complejas el concepto está inspirado en las redes sociales, representando círculos de amigos o conocidos en los cuales todos se conocen entre sí. Asumiendo un nodo i en una red compleja, con k_i aristas que lo conectan con k_i nodos. Si dichos nodos vecinos formaran parte de un cliqué, entonces habría k_i (k_i -1)/2 aristas entre sí. El coeficiente de clustering se define como la razón existente entre la cantidad real de aristas entre los nodos y la cantidad de aristas que lo harían un cliqué, es decir:

$$C_i = \frac{2E_i}{k_i(k_i - 1)}$$

El coeficiente de clustering de la red entera será el promedio de los C_i de cada nodo [Albert y Barabási, 2002].

El tercer concepto de las redes complejas es el de *degree distribution* o grado de distribución. Tal como se mencionó anteriormente, el grado de un nodo es la cantidad de arcos o aristas que posee. No todos los nodos en una red compleja poseen el mismo grado. La propagación del grado de nodos está definida por una función de distribución P (k) la que da una probabilidad de que un nodo cualquiera posea una determinada cantidad de aristas. En un grafo aleatorio el grado de distribución de un nodo será cercano al promedio <k> de dicha red y tendrá una distribución de Poisson con su pico en P (<k>). Para las redes complejas se ha demostrado que el grado de distribución varía marcadamente respecto a una distribución de Poisson, teniendo en muchos casos una distribución potencial del tipo $P(k) \sim k^{-y}$ [Albert y Barabási, 2002].

Las redes que cumplen con una función de distribución potencial del tipo $P(k) \sim k^{-y}$ son llamadas redes libres de escala (*scale-free*) ya que en ellas existe un pequeño conjunto de nodos, llamados *hub*, con un número de conexiones (arcos o aristas) mucho mayor al promedio [Wen *et al.*, 2007].

Figura 2.5 – Función de distribución potencial del tipo $\mathrm{P(k) \sim k^{-y}}$

Las redes libres de escala normalmente exhiben las siguientes propiedades [Wen *et al.*, 2009; Wen *et al.*, 2007):

1. Distribución potencial: el grado de distribución posee una cola decreciente
2. Hub: unos pocos nodos poseen muchas más conexiones que el promedio
3. Small-world: la distancia promedio entre dos nodos cualesquiera es pequeña comparada al tamaño de la red
4. Clustering: el coeficiente de agrupamiento es, para la mayoría de las redes complejas, superior al que se encontraría en una red aleatoria del mismo tamaño

5. Eficiencia de propagación: la información se transmite en una red compleja de una forma mucho más eficiente que en una red aleatoria gracias a la existencia de hubs [Pastor-Satorras y Vespignani, 2001].
6. Alta tolerancia a errores: una red compleja tiene una tolerancia muy superior a errores aleatorios que una red aleatoria
7. Vulnerable a ataques organizados: por su dependencia de los hubs, una red compleja es más vulnerable que una red aleatoria a ataques organizados

Estos nuevos conceptos, surgidos de diversas disciplinas, han permitido una transversalidad de conocimientos que hacen posible asumir la existencia de principios similares de orden en redes inicialmente diferentes como la Internet y el plegamiento de proteínas, ambas redes complejas libres de escala. La existencia de estos principios fue explicada por Barabási [Barabási, 2002] mediante un modelo de preferencias dinámicas, que indica que mientras más conexiones tenga un nodo, mayor será la probabilidad de que un nuevo nodo se conecte al mismo.

2.7.2 Sistemas software como redes complejas

Un sistema software puede ser tratado como una red compleja de componentes conectados por relaciones de dependencia, red a la que se denomina CDN (*Component Dependency Network* o red de dependencia de componentes), y que representa una vista de arquitectura del sistema software [Wen *et al.*, 2007].

El modelado de sistemas software como redes de dependencia de componentes (CDN), o como diagramas de colaboración de clases, en los cuales cada clase o paquete representa un nodo y cada llamada o conexión entre clases o paquetes representa un arco o arista, ha demostrado que los sistemas software son, en general, redes complejas [Wen *et al.*, 2007; Myers, 2003].

Myers [Myers, 2003] tras el análisis de diversos paquetes software como el sistema operativo Linux, el motor de bases de datos MySQL, el procesador de textos AbiWord y la librería de visualización VTK, entre otros, demostró de forma empírica que los sistemas software se comportan como redes complejas libres de escala. La investigación de Myers, realizada con grafos dirigidos demostró la existencia de grupos de nodos con muchas conexiones en comparación de los demás. Su análisis de los grados de distribución comprobó además que los sistemas tienen una distribución potencial del tipo $P(k) \sim k^{-y}$ tanto para conexiones de salida como para conexiones de entrada. En lo que respecta a la correlación de grados entre nodos, Myers concluyó que los componentes pueden poseer un alto grado de conexiones de salida (lo que implicaría una alta reusabilidad [Henderson-Sellers, 1995]) o un alto grado de conexiones de entrada [lo que implicaría excesiva complejidad [Henderson-Sellers, 1995]), correspondiéndose con componentes generadores de información y componentes

consumidores de información respectivamente. A partir de este análisis surgen dos conclusiones: la existencia de un componente con un alto grado de conexiones de salida y conexiones de entrada probablemente representará un problema en el diseño del sistema, asociado al concepto de complejidad fan-in fan-out y la existencia de una correlación positiva entre nodos con un alto grado de conexiones de salida los cuales tienden a estar conectados entre sí, situación que ocurre de forma similar en los nodos con un alto grado de conexiones de entrada. Por último, Myers demostró una clara tendencia de los vecinos de un nodo (los nodos que se encuentran conectado a este) a ser a su vez vecinos entre sí, lo que implicaría la existencia de agrupamiento entre nodos de similar grado. Este fenómeno ha sido estudiado por otros autores [Ravasz *et al.*, 2002] que concluyeron que un agrupamiento dependiente del grado de los nodos del tipo $C(k) \sim k^{-1}$ es un indicativo de sistemas jerárquicos.

En la figura 2.6 se puede visualizar una clase de colaboración del sistema VTK, estudiada por Myers, en donde se pueden apreciar varias características de las redes complejas libres de escala, como la existencia de hubs, la propiedad de mundo pequeño y el agrupamiento de nodos.

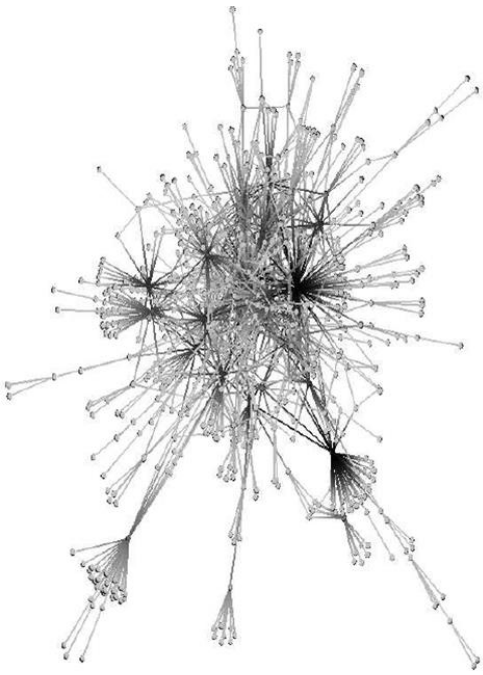


Figura 2.6 – Visualización de una clase de VTK como red compleja [Myers, 2003]

Wen *et al.* [Wen *et al.*, 2009; Wen *et al.*, 2007] han llegado a la misma conclusión de Myers, coincidiendo en que la mayoría de sistemas software se comporta como redes libres de escala, independientemente de su funcionalidad. Las pruebas realizadas por los autores sobre sistemas desarrollados en Java comprobaron que en los sistemas software, modelados como redes CDN, la desviación estándar del grado de distribución es aproximadamente tres veces el valor del grado promedio de todos los nodos del sistema, cuando en una red aleatoria debería ser igual. Además, el

coeficiente de agrupamiento es aproximadamente veinte veces mayor que el grado promedio de la red, cuando en una red aleatoria debería ser similar. Y por último, los autores comprobaron que todas las distribuciones de grado (de entrada, de salida y de ambas) muestran una función de distribución potencial.

El descubrimiento que revela que los sistemas software se comportan como redes complejas presenta una serie de ventajas teóricas y prácticas, entre las que podemos mencionar las siguientes [Wen *et al.*, 2009]:

- Es viable expresar que existe un conjunto de leyes que son la base para el desarrollo y la evolución de los sistemas software y de otras redes complejas como las sociales o biológicas.
- Gracias a la propiedad *scale-free*, se pueden identificar fácilmente componentes importantes en sistemas de software grandes.
- Un grado de distribución normal en un software podría indicar que el mismo se encuentra sobre desarrollado. Esta situación podría requerir un desarrollo anti regresivo.
- La tolerancia a errores en redes libres de escala permitiría comprender por qué errores aleatorios en sistemas software no siempre ocasionan fallos completos del sistema.

La demostración de que los sistemas software se comportan como redes complejas ha llevado al desarrollo de modelos específicos de redes complejas para su aplicación en sistemas software. Zheng *et al.* [Zheng *et al.*, 2008] desarrollaron dos modelos de evolución de software basados en la dependencia entre paquetes de Gentoo Linux.

El modelo de preferencias dinámicas, desarrollado por Barabási y mencionado previamente en esta sección, describe matemáticamente la probabilidad de que un nuevo nodo se conecte a nodos ya existentes mediante una preferencia lineal descripta por la función de probabilidad $\Pi(k) \sim k$ siendo k el grado del nodo existente. Esta función, para un *k* lo suficientemente grande, dará una función de distribución potencial del tipo $P(k) \sim k^{-y}$ en la cual *y* toma un valor de entre 2,1 y 4 para la mayoría de sistemas reales [Zheng *et al.*, 2008].

Krapivsky *et al.* [Krapivsky *et al.*, 2000] proponen una mejora al modelo de Barabási en el cual la preferencia no es lineal y depende de un valor ajustable, pasando a ser la probabilidad de que un nodo existente reciba una conexión por parte de un nodo nuevo $\Pi(k) \sim k^{\alpha}$. A su vez, Dorogovtsev y Mendes [Dorogovtsev y Mendes, 2002] proponen una mejora a ambos modelos en la cual la preferencia dinámica ya no depende únicamente del grado del nodo existente sino también de la edad (τ) del mismo. La función de probabilidad será entonces: $\Pi(k) \sim k\tau^{\beta}$ en donde β es un valor ajustable.

Zheng *et al*. observaron empíricamente que algunos componentes tienen una vida finita y en base a esta observación desarrollaron dos nuevos modelos de evolución de redes complejas: DDEA (*Degree Dependent adjustable Evolution with Aging* o Evolución ajustable dependiente de grado con antigüedad) y DAAE (*Degree and Age dependent Adjustable Evolution* o Evolución ajustable dependiente de grado y antigüedad). El modelo DDEA determina una probabilidad $\Pi(k) \sim k^{\alpha} e^{-\beta\tau}$ en donde $\tau = t - s_k$ siendo t el tiempo actual y s_k el tiempo de nacimiento del nodo *k* y los parámetros α y β son ajustables. Este modelo asegura que un nodo joven con un alto grado tiene una alta probabilidad de conectarse a un nuevo nodo, la cual disminuirá a medida que envejezca. El modelo DAAE, a su vez, determina una probabilidad $\Pi(k) \sim k^{\alpha} \tau^{-\beta k}$ con las mismas consideraciones que DDEA, pero dando a los nodos jóvenes de bajo grado una mayor probabilidad de conexión.

2.7.3 Métricas de software mediante redes complejas

Además de su uso para el modelado de sistemas software, las redes complejas también han sido utilizadas para el cálculo de métricas de software. Ma *et al.* [Ma *et al.*, 2005] desarrollaron un método cualitativo para la medición de la complejidad estructural en sistemas software basada en redes complejas. El método propuesto por Ma *et al.* mide la complejidad estructural del sistema a partir del concepto de entropía de Shannon [Shannon, 1949], definiendo como coeficiente de evaluación a $R = \sum_{i=1}^{n} \varphi_i r_i$ en donde r_i se define como $r = 1 - E/E_{max}$ y φ_i es el peso asociado a r_i tal que $\sum \varphi = 1$. E, asociado a la cantidad de incertidumbre, se define a su vez como $E = -\sum_{i=1}^{n} p_i \ln p_i$ y mide que tan homogéneo un set de componentes es partiendo de una serie de probabilidades (p_i) de una serie de valores variables, discretos y aleatorios q_i que representan la energía metamórfica de la estructura de un sistema.

El método descripto es cualitativo ya que para casos similares de R, se considera el peso de las aristas según la fórmula $W = (w_{max} - w_{min}) * \frac{\sum_{i=1}^{n} w_i}{n}$ en donde w es una función de la sumatoria de pesos que se asignan a una arista, la cual toma valores discretos en función de su tipo (por ejemplo, acceso a datos o llamada a procedimiento) y n es la cantidad total de aristas del sistema.

Solé y Valverde [Solé y Valverde, 2004] también han analizado sistemas software como redes complejas a partir de la entropía de los mismos y con el objetivo de determinar patrones de evolución comunes en el universo de la redes complejas. Los autores analizan la existencia de restricciones en la evolución de las redes complejas dentro de redes tecnológicas (incluyendo cuatro aplicativos software), redes biológicas y redes teóricas.

3. DESCRIPCIÓN DEL PROBLEMA

En este capítulo se describe la importancia de la medición a lo largo del proceso de desarrollo de software (sección 3.1), se identifica el problema de investigación a partir de las posibilidades de mejora detectadas en dicha área (sección 3.2), se caracteriza el problema abierto (sección 3.3) y se concluye con un sumario de investigación, en donde se presentan una serie de preguntas que se intenta responder mediante esta trabajo de tesis (sección 3.4).

3.1 Importancia de la medición en el proceso de desarrollo de software

La medición es uno de los aspectos más importantes en el proceso de desarrollo de software debido a que permite la comprensión de las características cuantitativas y cualitativas del software a lo largo de las diferentes etapas de desarrollo del mismo. El objetivo de la medición es proveer de información de valor para los procesos de toma de decisiones del ciclo de vida de un proyecto de software, tanto en lo que respecta a aspectos técnicos como a aspectos de gestión [Chhabra y Gupta, 2010].

La gestión de proyectos de desarrollo de software requiere de métricas para poder llevar a cabo de forma precisa el planeamiento y el control del proceso de desarrollo. Las métricas dan soporte a estas tareas ya que permiten administrar los procesos, productos y recursos relacionados con el proceso de desarrollo de software [Bellini *et al.*, 2008]. Los tres tipos de métricas descriptas en la sección 2.3 tienen incumbencia en estas tareas: las métricas de proceso ya que son necesarias para evaluar la eficiencia de un proceso y determinar aspectos como costos y tiempo de desarrollo; las métricas de proyecto en cuanto permiten evaluar el avance del proyecto y los recursos utilizados hasta el momento; y las métricas de producto, principalmente orientadas a atributos externos, dado que permiten evaluar factores del software como la calidad.

En lo que respecta a aspectos técnicos, las métricas permiten a los ingenieros evaluar diversos aspectos del desarrollo del software a lo largo de todas las etapas de su ciclo de vida [Chhabra y Gupta, 2010]. Las métricas de producto, tanto las orientadas a atributos internos del software como a atributos externos, son las que proveen el soporte a estas tareas permitiendo a los ingenieros evaluar aspectos del software como su confiabilidad, mantenibilidad y calidad.

Dentro del ciclo de vida del desarrollo de software es deseable poder realizar mediciones tan pronto como sea posible [Zhao *et al.*, 1998] dado que los errores producidos en los artefactos en etapas tempranas del desarrollo de software normalmente son propagados a etapas posteriores durante las cuales resultan más costosos de identificar y corregir [Boehm, 1981]. En particular, dentro de la fase de diseño de software se hace indispensable contar con métricas adecuadas para guiar el proceso de diseño [Pressman, 2005]. Las métricas en esta etapa permiten, de forma objetiva, identificar falencias

o posibilidades de mejora en el diseño además de predecir características externas del software como la reusabilidad y su facilidad de mantenimiento.

Asimismo, las métricas de software deberían capturar efectivamente el comportamiento real del software, aspecto que sólo puede medirse mediante las métricas dinámicas [Tahir y MacDonell, 2012]. Además, algunas características del paradigma orientado a objetos, como el polimorfismo y el uso de un sistema de tipos dinámicos, son también fundamentos para la elección de las métricas dinámicas teniendo en cuenta que las métricas estáticas han sido encontradas inadecuadas para la medición de dichas características [Chhabra y Gupta, 2010].

3.2 Identificación del problema de investigación

En la sección anterior se describió la importancia de la medición en el proceso de desarrollo de software, así como también las ventajas de las métricas de diseño y de las métricas dinámicas.

A los efectos de identificar el problema de investigación que se pretende resolver con la presente tesis se debe analizar el estado de las métricas de software en la actualidad. En los últimos años se han desarrollado diversos trabajos con el objetivo de analizar el panorama de métricas dentro de la ingeniería del software [Tahir y MacDonell, 2012; Chhabra y Gupta, 2010; Kitchenham, 2010]. En dichos trabajos se ha llegado a conclusiones similares, las cuales se describen a continuación:

- El estudio de métricas no está dominado por el paradigma orientado a objetos, el cual es el paradigma de desarrollo de software más utilizado en la actualidad y que representa aproximadamente un tercio de las métricas desarrolladas
- Existe una clara predominancia de las métricas estáticas por sobre las métricas dinámicas
- La cantidad de publicaciones sobre métricas dinámicas son cada vez más abundantes
- Las métricas desarrolladas, tanto estáticas como dinámicas, se concentran en general en atributos internos del software, como la cohesión y el acoplamiento, dejando de lado los atributos externos como la mantenibilidad. Esta diferencia se hace aún más marcada en las métricas dinámicas
- Dentro de las métricas dinámicas la gran mayoría está enfocada al código, situación que puede observarse en la figura 3.1. La reingeniería del software representa el segundo foco de interés dentro de las métricas dinámicas
- Las métricas con mayor impacto en la comunidad son aquellas validadas empíricamente

Esta serie de conclusiones permite afirmar que las métricas dinámicas de diseño, dentro del paradigma orientado a objetos, y, entendidas como aquellas que se enfocan en medir el

comportamiento en ejecución de aspectos del diseño de sistemas, no han tenido mayor desarrollo dentro del universo de métricas.

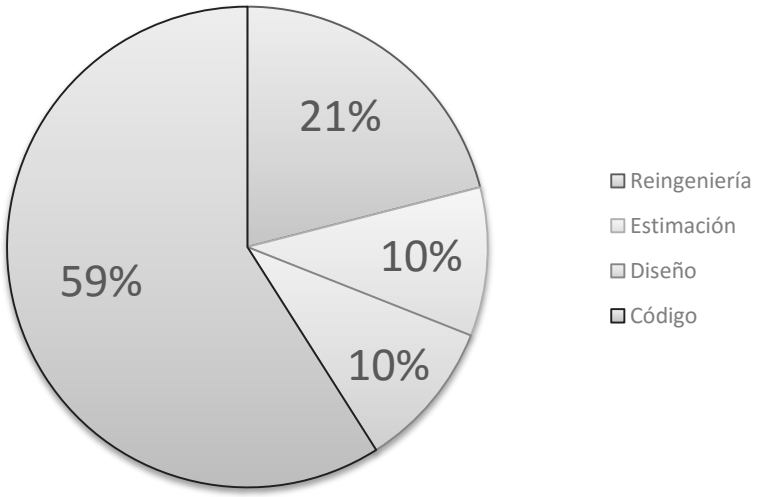


Figura 3.1 –Distribución de métricas de diseño por enfoque [Tahir y MacDonell, 2012]

3.3 Descripción del problema

En base a lo descripto en las secciones anteriores de este capítulo, es posible identificar un problema abierto, consistente en la necesidad de una suite de métricas dinámicas, aplicables en la etapa de diseño de sistemas e incluida dentro del paradigma orientado a objetos.

Una suite de métricas de estas características permitirá a los ingenieros del software contar con información precisa y oportuna. Precisa, ya que se ha demostrado la eficiencia de las métricas dinámicas para medir el comportamiento real del software y oportuna, ya que se obtendrá en una etapa previa a la codificación, pudiendo identificar y corregir posibles errores antes de la escritura del software. La suite de métricas, además, estará dirigida al paradigma orientado objetos, el cual concentra la mayor cantidad de desarrollos de software en la actualidad.

3.4 Sumario de investigación

Se propone en este trabajo de investigación el desarrollo de un conjunto de métricas dinámicas, definidas a nivel de diseño y aplicables dentro del paradigma de desarrollo de software orientado a objetos. Las métricas tendrán como objetivo brindar una mejor interpretación del funcionamiento y de las características del sistema en desarrollo y así servir como entrada para el proceso de toma de decisiones técnicas y de gestión del desarrollo de software.

El conjunto de métricas deberá ser aplicable dentro del proceso unificado de desarrollo de software y deberá respetar los lineamientos del lenguaje de modelado universal (UML). Las métricas, además, deberán adaptarse a la terminología de la norma internacional ISO/IEC 25000 y cumplir con los criterios de validación del estándar IEEE 1061.

Se propone una validación de las métricas por medio de un método empírico con el objetivo de responder a las siguientes preguntas de investigación:

Pregunta 1: ¿Es posible desarrollar un modelo de sistemas software basado en redes complejas, generable a partir de diagramas UML, basado en RUP, centrado en el diseño, y con el objetivo de calcular métricas sobre el mismo?

Pregunta 2: De ser posible desarrollar este modelo, ¿se puede generar un conjunto de métricas que estudien el comportamiento del sistema? En caso afirmativo, ¿qué métricas?

4. SOLUCION PROPUESTA

En este capítulo se propone una metodología de análisis del comportamiento de sistemas software, generada desde el punto de vista del diseño de sistemas, dentro del paradigma orientado a objetos y basada en redes complejas. Se especifican las generalidades de la metodología propuesta (sección 4.1) y se analiza en profundidad la estructura de la misma, describiendo en detalle sus actividades principales: el modelado del comportamiento del sistema mediante redes complejas (sección 4.2) y el cálculo de métricas orientadas al diseño de sistemas (sección 4.3).

4.1. Metodología de análisis dinámico del diseño de sistemas de información basado en redes complejas

En esta sección se presenta y describe una metodología de análisis dinámico del diseño de sistemas basada en redes complejas, denominada MADDS. En la sección 4.1.1 se analizan generalidades de la metodología, incluyendo su objetivo y sus características principales. Posteriormente en la sección 4.1.2 se realiza una descripción de la metodología, formalizando la propuesta de la misma y describiendo sus actividades principales. Finalmente en la sección 4.1.3 se analiza la integración de la metodología dentro del marco del proceso unificado de desarrollo.

4.1.1. Generalidades

En función del análisis realizado en el capítulo 3, correspondiente a la *Descripción del Problema*, se propone en este trabajo de tesis una metodología de análisis dinámico del diseño de sistemas basada en redes complejas y denominada MADDS. Esta metodología permitirá estudiar el comportamiento de un sistema a partir del diseño del mismo, permitiendo evaluar aspectos dinámicos del sistema sin necesidad de haber avanzado con la codificación o la etapa de pruebas, tal como se describió en el capítulo 2, *Estado de la Cuestión*. El uso de MADDS permitirá además, encontrar posibilidades de mejora o falencias en el diseño del sistema, lo que tendrá un impacto positivo en los costos de desarrollo.

La metodología propuesta en esta tesis (MADDS) presenta las siguientes características:

- Se integra dentro del proceso unificado de desarrollo, específicamente dentro del flujo de análisis y diseño
- Es compatible con las especificaciones de UML v2.x, teniendo su punto de partida en el Diagrama de Secuencia de UML, el cual modela la interacción entre los distintos componentes del sistema
- Está enfocada en el análisis de sistemas software bajo el paradigma orientado a objetos

- Contempla tanto una visión parcial del sistema como una visión integral del mismo, por lo que su aplicación permitirá estudiar desde una parte del diseño de un sistema hasta el diseño completo del mismo
- Su estructuración en actividades independientes permite la adaptación de la metodología a las necesidades de desarrollo. Cada uno de los procedimientos de MADDS podrá ser reemplazado por uno equivalente sin afectar la estructura de la metodología, pudiendo además agregar nuevos factores de análisis

4.1.2. Descripción de la metodología de análisis dinámico del diseño de sistemas basada en redes complejas (MADDS)

Tal como se describió en el capítulo 3, dentro del proceso unificado de desarrollo las tareas se dividen en flujos de trabajo, los cuales están insertos dentro de una o más fases, por tratarse de un proceso de desarrollo iterativo e incremental. MADDS se inserta dentro del flujo de Análisis y Diseño, que se lleva a cabo principalmente en la fase de elaboración (segunda fase del proceso unificado de desarrollo).

El flujo de trabajo del proceso unificado de desarrollo puede observarse en la figura 2.2 mientras que su modificación con MADDS puede observarse en la figura 4.1, pudiendo notarse que la salida del diseño de componentes es la entrada a MADDS. Cabe destacar que el flujo de trabajo representado corresponde a una única iteración dentro de RUP, por lo que cada una de las actividades se repetirá en caso de producirse iteraciones.

Tal como se describió en la sección 2.1, en donde se describieron las principales características del proceso unificado de desarrollo de software, cada una de las actividades representadas en los flujos de trabajo (figuras 4.1) consume y produce un conjunto de artefactos, los cuales son especificaciones de algunas características y/o atributos del sistema en desarrollo. De forma análoga a las demás actividades del proceso unificado de desarrollo, MADDS opera con artefactos, siendo su punto de partida el diagrama de secuencias de UML, artefacto que se genera en la actividad de diseño de componentes.

MADDS se encuentra dividido en dos actividades, que se describen esquemáticamente en la figura 4.2. A continuación se realiza una breve descripción de cada una de estas actividades:

1. La primera actividad consiste en el modelado del sistema como una red compleja y tiene como entrada el artefacto *diagrama de secuencias* generado en el diseño de componentes del proceso unificado de desarrollo. La salida de este proceso es un *modelo del sistema como red compleja*, lo que en términos de RUP debe ser descripto como un artefacto.

2. La segunda actividad es el cálculo de métricas orientadas al diseño y se inicia a partir del artefacto generado en la actividad anterior, arribando a la valuación de un conjunto de métricas que serán analizadas a fines de identificar errores y/o posibilidades de mejora en el diseño del sistema

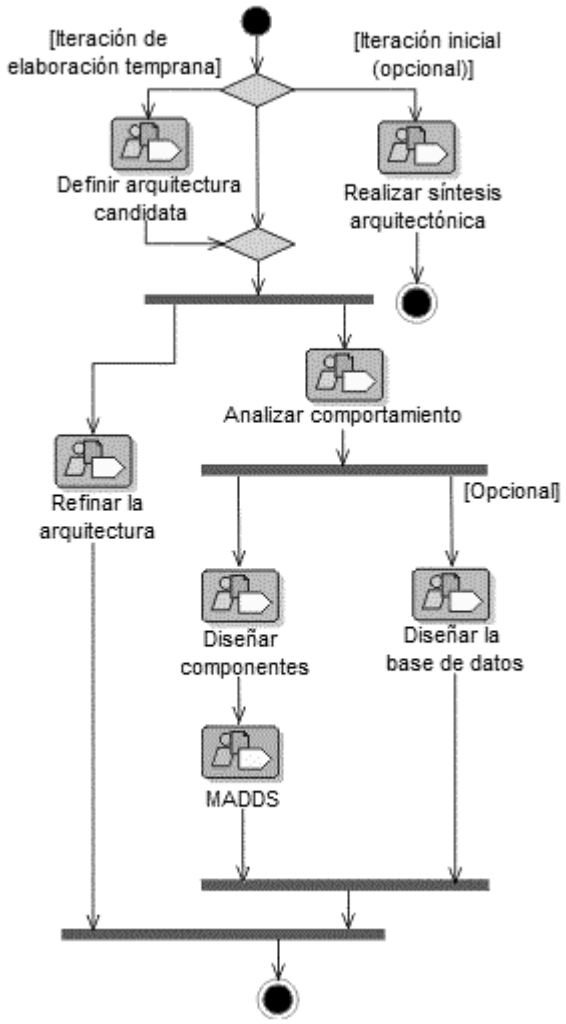


Figura 4.1 – Flujo de trabajo de RUP con MADDS para el Análisis y Diseño

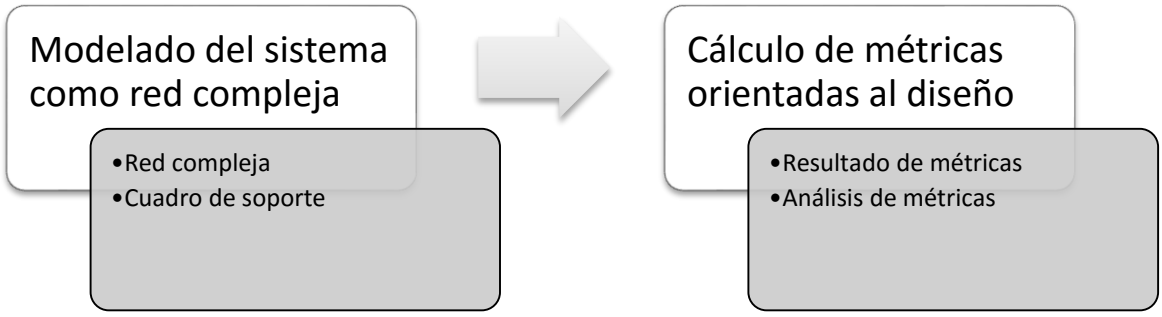


Figura 4.2 – Actividades de MADDS

4.1.3. Integración de MADDS dentro del proceso unificado de desarrollo

Uno de los objetivos propuestos para MADDS es que sea una metodología fácilmente adaptable a los desarrollos de software existentes. Visto y considerando que el proceso unificado de desarrollo, en conjunto con el lenguaje unificado de modelado (UML), constituyen el proceso y el lenguaje de representación más utilizados para el desarrollo de sistemas en la actualidad, MADDS se desarrolló de forma tal de adaptarse a ambos.

Ya observamos en la sección 4.1.2 tres de las características que permiten integrar MADDS al proceso unificado de desarrollo que son: su integración dentro del flujo de análisis y diseño, su división en actividades y la elaboración y el consumo de artefactos.

Por su parte, la actividad de modelado del sistema como red compleja parte del diagrama de secuencia de UML. Si bien UML es a su vez un estándar dentro del proceso unificado, es importante destacar el motivo de la elección de los diagramas de secuencia como los artefactos de entrada para MADDS.

Dentro de los diagramas de comportamiento (los únicos que consideran aspectos dinámicos del sistema en estudio) se decidió trabajar con diagramas de interacción ya que determinan una serie de criterios mediante los cuales queda documentado en el modelo toda relación dinámica entre las partes del sistema. Dentro de los diagramas de interacción, a su vez, el más utilizado es el diagrama de secuencia ya que en el mismo se observan prácticamente todos los aspectos dinámicos del sistema, incluyendo todas las interacciones entre clases, objetos y actores.

En los diagramas de secuencia, además, se pueden modelar cuestiones temporales de la interacción entre objetos debido a que se debe modelar el nacimiento y muerte de los mismos. Además de estas características, la elección de los diagramas de secuencia se fundamenta en las siguientes cuestiones:

- Los diagramas de secuencia se realizan normalmente en todos los desarrollos y en particular en casos de uso complejos
- Los diagramas de secuencia permiten observar todo el ciclo de vida de los objetos (desde su creación hasta su destrucción) en el eje vertical y de todos los mensajes que intercambian con otros objetos en el eje horizontal
- Si bien cada diagrama de secuencia modela un solo caso de uso del sistema, la sumatoria de los casos de uso da como resultado todas las interacciones en las cuales se puede ver involucrada una clase u objeto

Tal como se describió en la sección 4.1.2, MADDS se encuentra dividido en dos actividades, cada una de las cuales se relaciona con el ambiente a través de artefactos. La relación entre los artefactos de MADDS y los del proceso unificado se describe en la figura 4.3.

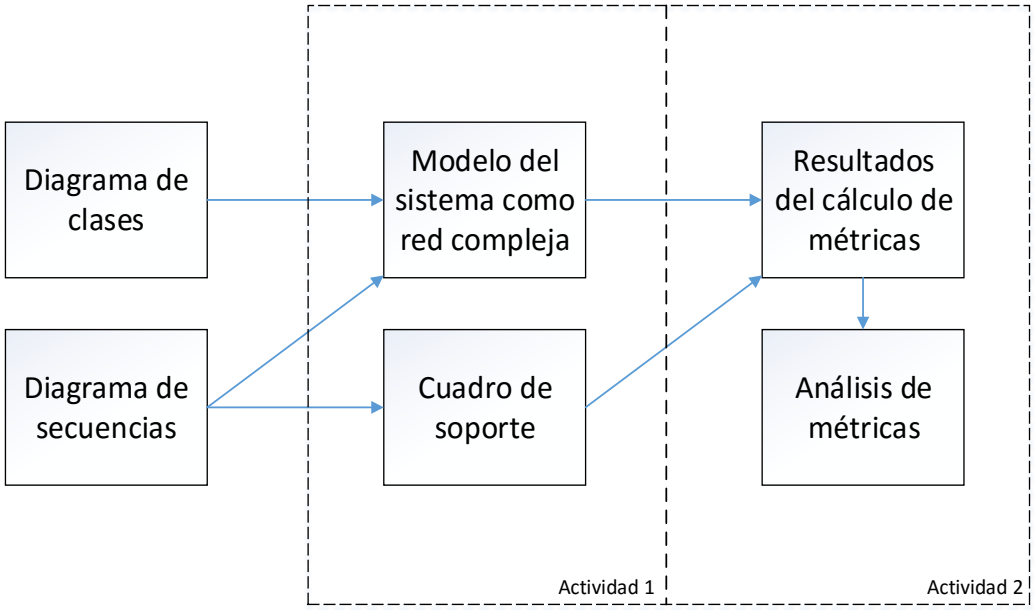


Figura 4.3 – Relación entre artefactos de MADDS

Tal como puede observarse en la figura 4.3, MADDS parte de dos artefactos del proceso unificado que son comunes en prácticamente todos los desarrollos de software de la actualidad: el *diagrama de clases* y el *diagrama de secuencias*. Ambos son utilizados en la primera actividad, en la cual se generan a su vez otros dos artefactos: el *modelo del sistema como red compleja* y un *cuadro de soporte* del mismo. La segunda actividad consume estos artefactos para la generación del artefacto *resultados del cálculo de métricas* a partir del cual se desarrolla el último artefacto de la metodología: *el análisis de métricas*.

4.2 Modelado del comportamiento del sistema mediante redes complejas

En esta sección se presenta un método de modelado de diagramas de secuencia de UML como una red compleja, el cual se implementa en la primera actividad de MADDS, y partir del cual se realizan todos los cálculos de métricas para el estudio dinámico del diseño de sistemas.

Es menester destacar que el resultado de la aplicación de este proceso de modelado excede los conceptos de una red compleja dado que no solo está formado por elementos propios de la teoría de grafos sino que además incluye otros elementos de soporte, necesarios para el cálculo de las métricas en la segunda actividad de MADDS. La relación que va a existir entre los componentes del diagrama de secuencia y los componentes de la red compleja van a estar dados por los indicados en la tabla 4.1.

Diagrama de secuencia	Red compleja
Objeto	Nodo o vértice
Mensaje o método	Arista o arco dirigido

Tabla 4.1 – Relación entre componentes del diagrama de secuencia y la red compleja

Además de una identificación del método, cada método tendrá dos valores asociados: un peso que será determinado por su complejidad algorítmica y un número que indique la cantidad de veces que se ejecutó.

Tal como se describió en la sección 4.1.3, el proceso de modelado parte de un diagrama de secuencia. A fines de facilitar la comprensión de la aplicación de este proceso, a lo largo de la presente sección se utilizará como ejemplo el diagrama de la figura 4.4. Este diagrama de secuencia, tomado de un escenario real [UPM, 2012], representa un caso de uso genérico para venta de entradas a espectáculos.

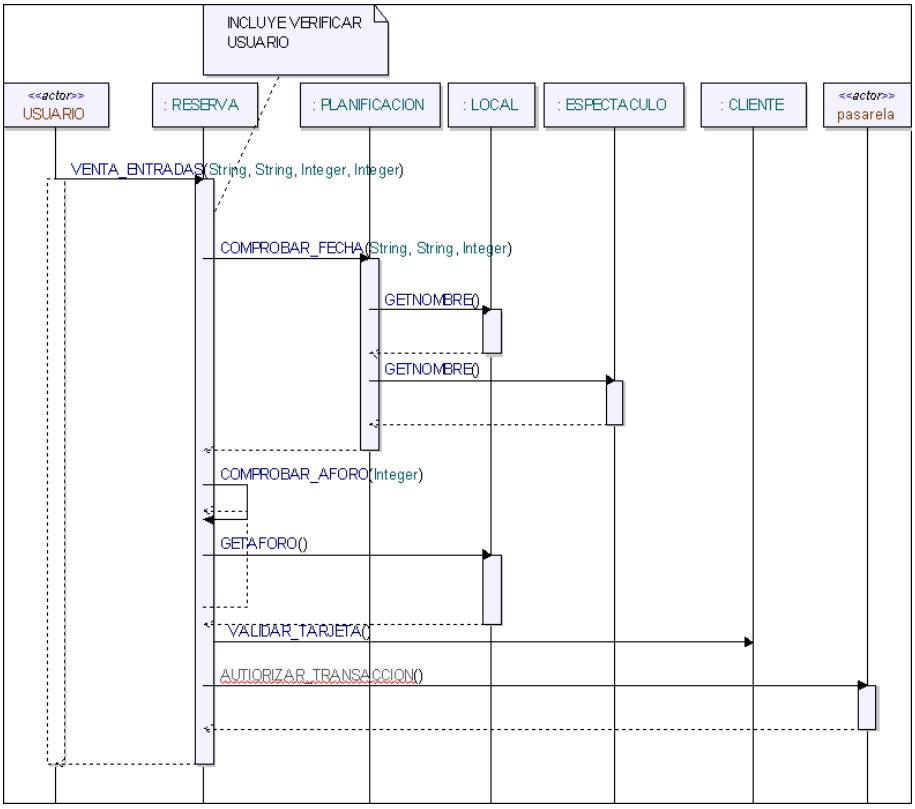


Figura 4.4 – Diagrama de secuencia UML

A partir del diagrama de secuencia del caso de uso a modelar, se debe proceder de la siguiente manera:

a) *Convertir todas las clases del diagrama en nodos*. Las clases se identifican en el diagrama como un rectángulo con el nombre de la clase precedido por un signo de dos puntos (:). En lugar de tomar las clases del diagrama de secuencias, también es factible modelarlas a partir del diagrama de clases. La ventaja de utilizar el mismo será que desde un primer paso se tendrá una visibilidad total de las clases del sistema. Es desaconsejable el uso del diagrama de clases en este paso en caso de que solo se vaya a estudiar una parte del sistema. En el diagrama de secuencias, debajo de cada clase se traza una línea conocida como línea de vida, en la cual se determinan los momentos de nacimiento y muerte de los objetos de dicha clase. En caso de que una clase tenga n instanciaciones simultáneas en el curso del caso de uso en estudio,

existirán n rectángulos con dicha clase. Solo es necesario modelar un único nodo en MADDS. El resultado de este paso puede observarse en la figura 4.5.

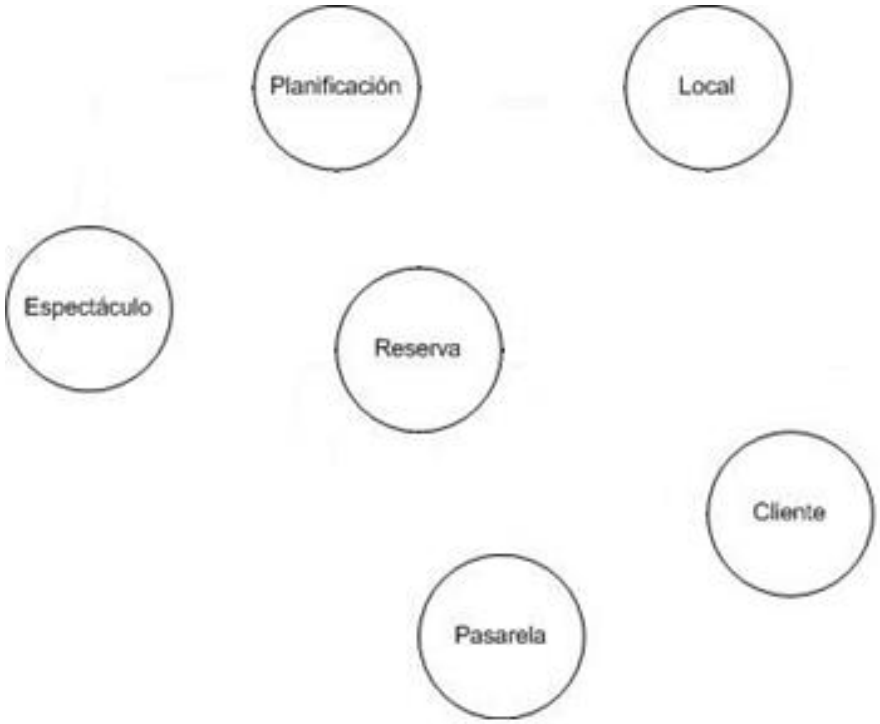


Figura 4.5 – Resultado del primer paso de MADDS

b) *Armar el cuadro de soporte para el cálculo de métricas*. Se debe generar un cuadro de cinco columnas indicando en la primera el nombre de clase y en la segunda cada uno de los métodos que dicha clase posee. Esta información, de forma análoga al paso anterior, puede ser tomada a partir del diagrama de clases, considerando únicamente las clases que se hayan modelado hasta el momento. Para cada tipo de llamada o parametrización diferente que tenga un método se deberá considerar una nueva fila en la tabla, debido a que en esta metodología resultante relevante estudiar el polimorfismo y la sobrecarga de los mismos. Al finalizar, hay que asignarle a cada método un identificador en la tercera columna, sugiriéndose una letra del alfabeto. Como dato adicional hay que agregar un asterisco a cada método que sea heredado de una clase padre, información que puede obtenerse del diagrama de clases correspondiente y que habría que asentar junto al identificador asignado al método. La salida de este paso para el ejemplo en análisis puede observarse en la tabla 4.2:

Clase	Método	Id método	Cantidad de llamadas (α)	Complejidad
Planificación	ComprobarFecha(Str,Str,Int)	A		
Local	GetNombre()	B		

	GetAforo()	E		
Espectáculo	GetNombre()	C		
Reserva	ComprobarAforo(Int)	D		
Cliente	ValidarTarjeta()	F		
Pasarela	AutorizarTransacción()	G		

Tabla 4.2 – Resultado del segundo paso de MADDS

c) *Contabilizar llamadas a los métodos (α)*. Se debe proceder analizando de forma visual el diagrama de secuencias y contabilizando, clase por clase, las flechas entrantes en la Línea de Vida. Cada flecha entrante representará una llamada a un método de dicha clase y a partir de cada llamada se deberá proceder sumando un número en la línea correspondiente del cuadro armado en el paso (b). Además, se deberá trazar una arista dirigida entre el nodo clase origen del mensaje y el nodo clase destino en el grafo armado en el paso (a), rotulando dicha arista con la nomenclatura seleccionada en el punto (b). De los tres tipos de mensajes descriptos en la sección 2.2 solo es necesario procesar aquellos mensajes que sean llamadas asíncronas (línea completa). Las salidas de este paso pueden verse en la figura 4.6 y en la tabla 4.3.

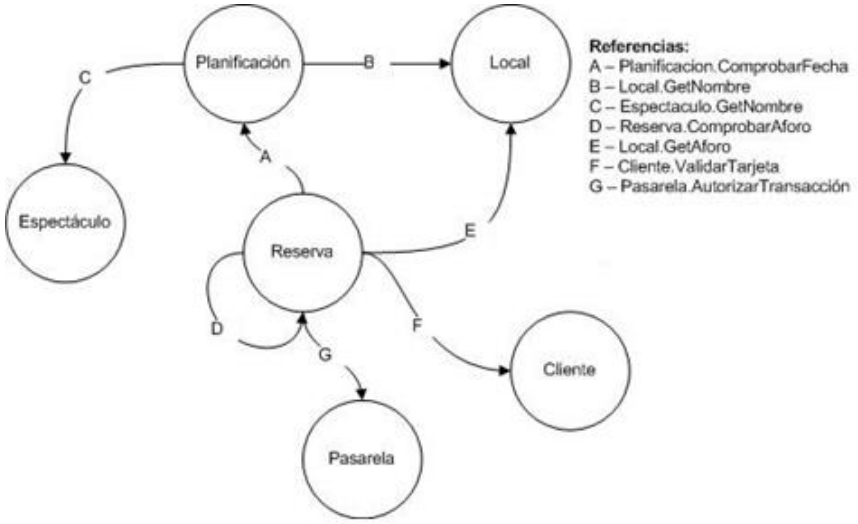


Figura 4.6 – Resultado del paso (c) de MADDS

Clase	Método	Id método	Cantidad de llamadas (α)	Complejidad
Planificación	ComprobarFecha(Str,Str,Int)	A	1	
Local	GetNombre()	B	1	
	GetAforo()	E	1	
Espectáculo	GetNombre()	C	1	
Reserva	ComprobarAforo(Int)	D	1	
Cliente	ValidarTarjeta()	F	1	

Pasarela	AutorizarTransacción()	G	1	

Tabla 4.3 – Resultado del paso c de MADDS

d) *Calcular la complejidad de cada método*. Para el cálculo de complejidad del método es factible utilizar cualquier métrica de complejidad orientada a la complejidad algorítmica o computacional. En esta tesis utilizaremos la métrica de complejidad ciclomática definida por McCabe [McCabe, 1976]. Esta métrica ha sido validada en decenas de oportunidades y es la base para varias métricas analizadas en el estado del arte, como las métricas de Yacoub *et al*. El cálculo de la complejidad ciclomática está dado por:

$$CC = 1 + ifs + loops + cases$$

En donde:

- *ifs* es la cantidad de operadores IF en el código
- *loops* es la cantidad de bucles (WHILE, FOR y REPEAT, en la mayoría de los lenguajes)
- *cases* es la cantidad de ramas de switching existentes, sin considerar el default

Clase	Método	Id método	Cantidad de llamadas (α)	Complejidad
Planificación	ComprobarFecha(Str,Str,Int)	A	1	3
Local	GetNombre()	B	1	7
	GetAforo()	E	1	9
Espectáculo	GetNombre()	C	1	7
Reserva	ComprobarAforo(Int)	D	1	11
Cliente	ValidarTarjeta()	F	1	6
Pasarela	AutorizarTransacción()	G	1	6

Tabla 4.4 – Resultado del paso d de MADDS

e) *Completar el modelo del sistema como red compleja*. Se debe proceder a rotular en cada arista la cantidad de llamadas que recibe el método al cual representa, obteniendo dicha información del cuadro generado en el punto anterior. La salida de este paso, y resultado final del proceso de modelado, puede observarse en la figura 4.7.

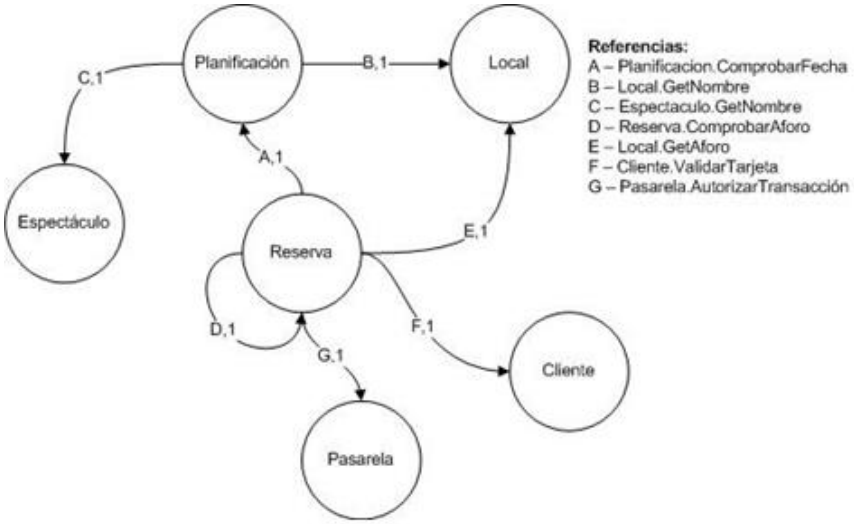


Figura 4.7 – Resultado final del subproceso de modelado de MADDS

El cuadro generado en el paso (c) y el grafo completado en el paso (d) pasan a considerarse dos artefactos de RUP y son la entrada al segundo subproceso de MADDS, en el cual se calculan las métricas que evalúan la dinámica del diseño de sistemas.

Cabe señalar que los resultados obtenidos a lo largo del ejemplo solo corresponden a un único caso de uso, y tal como se señalara precedentemente, para una mejor evaluación del diseño del sistema se debe realizar el análisis sobre la mayor cantidad (y de ser posible todos) los casos de uso. Seguidamente, y continuando con la numeración de los pasos anteriores, se describe el proceso de adición lógica de cuadros y grafos para el caso de analizar varios casos de uso.

f) *Generar el análisis para cada caso de uso que se quiera estudiar*, tal como se describió entre los pasos (a) y (d)
g) *Realizar la suma de los grafos.* La suma de grafos consiste en realizar una unión entre los grafos obtenidos, realizando la suma de la cantidad de llamadas identificadas en cada arista. A tal fin se sugiere proceder de la siguiente forma:
    i. Identificar todos los nodos involucrados y graficarlos. En el caso de que en el paso (a) se hayan modelado todas las clases, no es necesario realizar la identificación de los nodos.
    ii. Tomando un primer grafo de un caso de uso, pasar todas las aristas al nuevo grafo con todos los nodos. Pasar además de las aristas, su correspondiente nomenclatura y la cantidad de llamadas que tiene el método.
    iii. Para los siguientes grafos ir pasando las nuevas aristas que no existan en el grafo con su correspondiente nomenclatura y cantidad de llamadas. En caso de que la arista

(fácilmente distinguible por el Id) ya exista, se deberá adicionar la cantidad de llamadas.

h) *Realizar la suma de las tablas.* Para lo cual se deberá proceder de la siguiente forma:
   i. Generar una tabla base con la totalidad de los nodos (o clases del sistema), cada uno con todos sus métodos listados y los identificadores utilizados
   ii. Recorrer cada tabla e ir sumando en la tabla base la cantidad de llamadas de cada método

El resultado final consistirá de un grafo y una tabla con la totalidad de los casos de uso estudiados. En las figuras 4.8a y 4.8b pueden observarse dos casos de uso cuya adición daría como resultado el grafo de la figura 4.7.

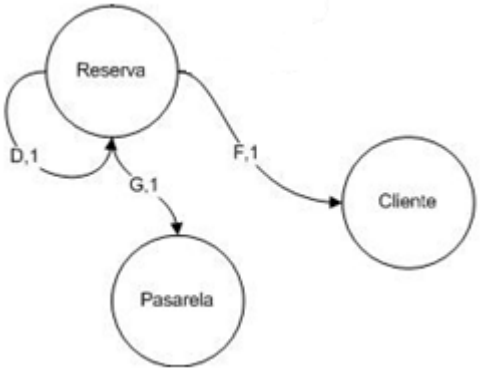


Figura 4.8a – Caso de uso A

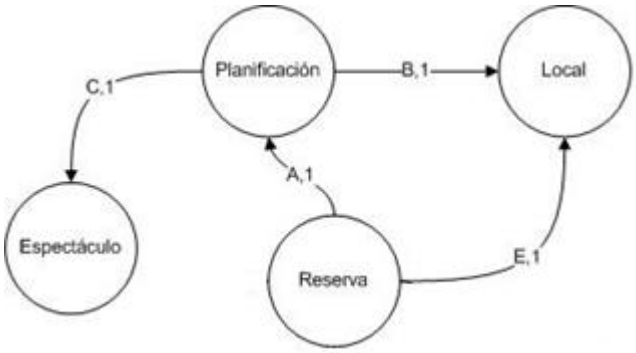


Figura 4.8b – Caso de uso B

4.3. Cálculo de métricas orientadas al diseño de sistemas

A partir del modelo generado en la sección anterior se presenta en esta sección un conjunto de métricas que permiten estudiar el diseño de un sistema desde un punto de vista dinámico. Las métricas son desarrolladas en dos secciones: métricas propuestas en esta tesis (sección 4.3.1) y métricas existentes en otros dominios, reinterpretadas al dominio de sistemas (sección 4.3.3).

4.3.1 Métricas propuestas en esta tesis

En la esta sección se presenta un conjunto de métricas que fueron desarrolladas para la presente tesis con el objetivo de caracterizar diversos aspectos del diseño de sistema.

4.3.1.1 Cantidad de nodos

La cantidad de nodos posiblemente sea uno de los principales indicadores del diseño de un sistema debido a que determina la totalidad de clases, y por ende, el tamaño del sistema. El tamaño de un sistema es un indicador básico de su complejidad.

Más allá de determinar el tamaño de la red compleja (y de forma transitiva, del sistema), la utilidad real de la cantidad de nodos es servir como factor de ponderación para el resto de las métricas de forma tal de permitir la proporcionalidad entre las mismas a la hora de comparar sistemas entre sí.

La cantidad de nodos se puede calcular de forma muy sencilla en el gráfico contando todos los nodos presentes. Tomando como base la tabla el conteo es igual a la cantidad de filas de la primera columna.

De forma numérica la cantidad de nodos, u orden del grafo, se representa como *V*.

4.3.1.2 Cantidad de aristas

La cantidad de aristas representa la cantidad de mensajes o métodos totales que se intercambian en el sistema. Tiene atributos similares a la métrica de *cantidad de nodos*. Resultan menester considerar que dentro de la aplicación de MADDS cada arista representa la llamada a un método con un conjunto de parámetros en particular.

De forma análoga a la *cantidad de nodos*, esta métrica permite medir el tamaño y, de una forma básica, la complejidad de un sistema. Es también útil como factor de ponderación para otras métricas.

El cálculo esta métrica puede realizarse contando en el grafo la cantidad de flechas existentes o de forma más sencilla, contando la cantidad de filas de la columna B del cuadro de soporte.

De forma numérica la cantidad de aristas, o tamaño del grafo se representa como *E*.

4.3.1.3 Llamadas totales

Esta métrica representa la cantidad total de invocaciones o llamadas que se realizan a los diferentes métodos durante la ejecución del sistema en análisis. Permite medir de forma absoluta la cantidad de mensajes que se intercambian en el sistema, valor que puede ser asociado con el tamaño y la complejidad del sistema.

La cantidad de llamadas a métodos puede calcularse sumando todas las llamadas modeladas en el grafo (α) o a partir de sumatoria de la última columna del cuadro de soporte.

De forma numérica se define como $LL = \sum \alpha$

4.3.1.4 Promedio de llamadas por método

El promedio de llamadas por método permite analizar la cantidad de veces, en promedio, que un método es ejecutado en un sistema. Es una forma de medir la carga promedio que tienen los métodos en el sistema.

Se define numéricamente como $LL(M) = \frac{LL}{E}$

4.3.1.5 Promedio de llamadas por clase

El promedio de llamadas por clase permite analizar la proporción entre llamadas y cantidad de clases. Es una forma de medir la carga promedio que tienen las clases en el sistema.

Se define numéricamente como $LL(V) = \frac{LL}{V}$

4.3.1.6 Métodos sin uso

No debería existir en un sistema un método que no se utilice y desde luego, es útil identificarlos en etapas tempranas para evitar que se desarrolle código que posteriormente no se utilizará. Un método de estas características desde luego tampoco podría pasar un test de integración ya que no se ejecutaría nunca.

La diferencia entre la cantidad de aristas que ingresan a un nodo en la red compleja y la cantidad de métodos de dicho nodo debería ser nula. Un valor positivo implicará que se está invocando a un método inexistente (lo que significa un error de procedimiento en el proceso descripto en la sección 4.2) y un valor negativo implicará la existencia de métodos que no se utilizan en el sistema. Es importante recordar que dentro de MADDS se considera cada llamada a un método como un método diferente, por lo que esta métrica determinará no sólo la existencia de métodos sin uso sino también de posibles tipos de llamadas a métodos que no se encuentren en uso.

Es posible obtener esta métrica a partir del cuadro generado en la sección 4.2, para dicho cálculo se debe contar la cantidad de filas de la columna B en donde la cantidad de llamadas en la tercera columna sea igual a cero.

Se define numéricamente como M_0

4.3.1.7 Métodos con un único uso

Es interesante conocer la cantidad de métodos de un sistema que a lo largo de todos los escenarios de ejecución se utiliza una sola vez. Su identificación permitirá evaluar posibilidades de mejora en el diseño del sistema. Al igual que en el caso de la métrica anterior, resulta primordial recordar que cada tipo de llamada polimórfica a un método es considerada como un método diferente en MADDS.

Los métodos con un único uso se pueden identificar en el modelo del sistema como aquellas aristas que tienen un uno como α mientras que en la tabla serán todas aquellas filas que tengan un uno en cantidad de llamadas (cuarta columna).

Se define numéricamente como M_1

4.3.1.8 Métodos muy utilizados

El hecho de que existan métodos que tengan muchas llamadas normalmente implicará la existencia de un hub, fenómeno que es analizado a nivel de clases y no de métodos y que se relaciona íntimamente con la cohesión y el acoplamiento. La métrica de métodos muy utilizados se limita al análisis de un conjunto de métodos que estadísticamente tiene un uso notablemente superior a los demás.

Es fundamental conocer aquellos métodos muy utilizados ya que tendrán un claro impacto en la ejecución del sistema. Además, se debe conocer de qué forma se distribuye la carga de trabajo en el universo de métodos del sistema dado que a partir de la misma se pueden definir grados de reusabilidad, cohesión y acoplamiento.

El cálculo de métodos muy utilizados considera a aquellos métodos que tienen al menos un 50% más de llamadas que el promedio de llamadas del sistema. De forma numérica, el umbral (α_μ) a partir del cual se considerarán métodos estará dado por $\alpha_\mu = 1{,}5 * LL(M)$

Todos aquellos métodos cuya cantidad de llamadas (α) sea igual o mayor al umbral α_μ definido, serán contabilizados como métodos muy utilizados (M_μ). Interesa conocer qué porcentaje de métodos, en relación a la cantidad total de métodos existentes, tiene mucho uso. Estos estarán dados por $\mu = \frac{M_\mu}{E} * 100$.

Un porcentaje alto de métodos muy utilizados probablemente indicará un problema de diseño en el sistema considerando que indicará un alto acoplamiento. Un porcentaje bajo, en cambio, indicará un bajo acoplamiento y, probablemente, una alta cohesión. Es posible que esta métrica tome el valor de cero para el caso de que no haya métodos con una cantidad de llamadas al menos un 50% superior al promedio, caso en el cual podrá adaptarse el umbral definido.

4.3.1.9 Tendencia central de la complejidad

Una de las características más interesantes del software, y sobre la cual se han desarrollado mayor cantidad de métricas, es la complejidad del software. La complejidad tiene varias implicancias dentro de la ingeniería del software pero en esta sección se considera como la dificultad inherente del cálculo computacional.

Dentro de la sección 4.2 se describió la forma de cálculo de la complejidad para los diferentes métodos utilizando una métrica estática. Interesa conocer el peso de dicha complejidad en la dinámica del sistema para lo cual se ponderará la misma con la cantidad de ejecuciones del método.

La tendencia central de la complejidad será evaluada utilizando el promedio y la desviación estándar, además de la mediana. Para el cálculo del promedio es posible aplicar la propiedad distributiva y evaluar la tendencia central a nivel de clase en lugar de sistema.

Para todo el sistema, el promedio de la complejidad dinámica será:

$$CP = \frac{\sum_1^E (C_E * \alpha_E)}{E}$$

En donde:

- CP es la complejidad dinámica promedio
- C_E es la complejidad del método E
- α_E es la cantidad de llamadas del método E
- E la cantidad total de aristas o métodos del sistema

Interesa conocer la desviación estándar (DCP) de la complejidad dinámica de los métodos con respecto a la CP para saber cómo se agrupan en torno al mismo. A tal fin se debe proceder con el cálculo:

$$DCP = \sqrt{\frac{1}{E}\sum_1^E ((C_E * \alpha_E) - CP)^2}$$

En los cálculos anteriores se tomó como referencia a la cantidad de métodos E para el cálculo del promedio. Además de este cálculo, es posible conocer la complejidad ponderada por la cantidad de clases del sistema (o nodos en el grafo complejo). Para este caso, el cálculo de la complejidad dinámica promedio estará dado por:

$$CP' = \frac{\sum_1^E (C_E * \alpha_E)}{V}$$

En donde:

- CP’ es la complejidad dinámica promedio por clase
- C_E es la complejidad del método E
- α_E es la cantidad de llamadas del método E
- V la cantidad total de nodos o clases del sistema

El promedio, o media aritmética, se ve influenciado por los extremos, situación que se ve remarcada en el análisis del software en donde es usual que haya métodos de extrema baja complejidad o métodos de extrema alta complejidad. En el análisis dinámico esta situación se agrava ya que también es usual la existencia de métodos o clases que reciben un muy alto, o muy bajo, número de llamadas. Por este motivo es que dentro de esta métrica se calcula también la mediana, la cual estadísticamente no se ve influenciada por los extremos.

La forma de cálculo de la mediana es más compleja que la del promedio, se debe proceder obteniendo todos los *n* valores $m = C_E * \alpha_E$ ordenados de forma creciente y posteriormente identificando aquel cuya posición sea central. Es decir que la mediana de la complejidad estará dada por $MC = m_{(n+1)/2}$ para un *n* impar y $MC = (m_{n/2} + m_{(n/2)+1})/2$ para un *n* par.

4.3.1.10 Complejidad dinámica total del sistema

La complejidad dinámica del sistema estará dada por la sumatoria de los productos de la complejidad de los métodos por la cantidad de ejecuciones que tienen los mismos. Esta métrica define de una forma básica la complejidad total de un sistema en ejecución y, al igual que la métrica anterior, está asociada al esfuerzo inherente al cálculo computacional.

Su cálculo estará dado por:

$$CDS = \sum_{1}^{E} (C_E * \alpha_E)$$

En donde:

- CDS es la complejidad dinámica total del sistema
- E es la cantidad total de aristas o métodos del sistema
- C_E es la complejidad del método E
- α_E es la cantidad de llamadas que recibe el método E

4.3.1.11 Cohesión

En la sección 2.4.3, métricas dinámicas de cohesión, se describió el concepto de cohesión y las métricas más significativas que estudian dicha característica. La cohesión es una característica deseable en un sistema ya que se busca que todas las partes de una clase (métodos y variables) estén relacionadas entre sí. Existen distintos tipos de cohesión, sin embargo, en esta métrica se considerará la cohesión funcional, normalmente considerado el tipo de cohesión más deseado.

La medición dinámica de la cohesión se realizará a partir del diagrama de clases y del cuadro de soporte desarrollado previamente, requiriendo además conocer los atributos privados de la clase con

los que interactúa cada método. Es factible que en una primera iteración del diseño no se posea esta información, caso en el cual esta métrica no podrá calcularse. Es necesario conocerlo para poder medir la cohesión funcional de una clase, ya que consideramos que un método no es cohesivo cuando en su ejecución no utiliza ninguna de las variables privadas de la clase a la que pertenece. Un método con estas características podría implementarse en otra clase.

El cálculo de esta métrica parte de la identificación de los n métodos no cohesivos pertenecientes a cada clase a partir de los cuales se puede calcular la cohesión funcional de la misma, considerando que la cohesión de la clase estará dada por la sumatoria de las llamadas a métodos no cohesivos, dividido la cantidad total de llamadas a métodos de clase. De forma numérica:

$$G_C = \frac{\sum_1^E \alpha_{ENC}}{\sum_1^E \alpha_E}$$

En donde:

- G_C es la cohesión de una clase C determinada
- E es la cantidad de métodos de la clase C
- α_{ENC} es la cantidad de llamadas a métodos no cohesivos
- α_E es la cantidad total de llamadas a métodos de la clase

Obteniendo las diferentes cohesiones de cada clase se podrá obtener la cohesión total del sistema (G) mediante el promedio de todas las G_C, es decir $G = \frac{\sum G_C}{C}$, en donde C es la cantidad de clases del sistema.

4.3.1.12 Polimorfismo

La métrica de polimorfismo determina la proporción entre las llamadas a métodos polimórficos y las llamadas totales del sistema. Define de esta forma la importancia del polimorfismo en la ejecución del sistema debido a que determina su incidencia en el sistema en ejecución.

A partir del cuadro de soporte se pueden identificar aquellos métodos polimórficos como aquellos que poseen los mismos atributos pero de diferente tipo. La sumatoria de las llamadas a métodos polimórficos dividido por la cantidad de llamadas total del sistema será:

$$P = \frac{\sum_1^E \alpha_{EP}}{\sum_1^E \alpha_E}$$

En donde:

- P es el polimorfismo total del sistema

- E es la cantidad de métodos del sistema
- α_{EP} es la cantidad de llamadas a métodos polimórficos
- α_E es la cantidad total de llamadas a métodos

4.3.1.13 Sobrecarga

La sobrecarga de un método se produce cuando un método puede ser llamado mediante el uso de diferentes parámetros. De forma intuitiva pueden identificarse en el cuadro de soporte como aquellos que tienen el mismo nombre pero diferentes parámetros. La métrica de sobrecarga dinámica, de forma análoga a la métrica de polimorfismo, calcula la proporción de llamadas a métodos sobrecargados por sobre la cantidad de llamadas total del sistema. De forma numérica:

$$S = \frac{\sum_1^E \alpha_{ES}}{\sum_1^E \alpha_E}$$

En donde:

- S es la sobrecarga total del sistema
- E es la cantidad de métodos del sistema
- α_{ES} es la cantidad de llamadas a métodos sobrecargados
- α_E es la cantidad total de llamadas a métodos

4.3.1.14 Acoplamiento

El acoplamiento en un sistema puede ser evaluado desde distintos puntos de vista ya que las clases en un sistema se relacionan de múltiples formas; es posible que por ejemplo las clases llamen a métodos de otras clases, que las clases hereden funcionalidad o atributos de una clase padre o que las clases accedan a atributos de otras clases. Algunas de las métricas ya analizadas en esta sección permiten medir el acoplamiento. Por ejemplo, la métrica de promedio de llamadas por clase es un indicativo del acoplamiento promedio del sistema desde el punto de vista de las llamadas entre clases.

Esta métrica tiene como objetivo identificar aquellas clases con un elevado acoplamiento, el cual podrá ser de acoplamiento de salida o acoplamiento de entrada, de forma análoga a las métricas desarrolladas por Arisholm vistas en la sección 2.4.2.2. Consideraremos que una clase tiene un alto acoplamiento de salida cuando realiza al menos un 50% más de llamadas que el promedio de todas las clases y tendrá un alto nivel de acoplamiento de entrada cuando reciba un 50% más de llamadas que el promedio de todas las clases. Al igual que en el caso de la métrica de métodos muy utilizados, este parámetro podrá ser modificado en función de las necesidades del sistema.

El umbral, igual para ambos casos, estará dado por $\alpha_\psi = 1{,}5 * LL(V)$. Todas aquellas clases que realicen más de α_ψ llamadas tendrán un alto nivel de acoplamiento de salida (ψ_S) y todas aquellas clases que reciban más de α_ψ llamadas tendrán un alto nivel de acoplamiento de entrada (ψ_E). Para ambos casos interesa conocer el porcentaje total de clases sobre el total del sistema:

$$A_S = \frac{\psi_S}{V} * 100$$

$$A_E = \frac{\psi_E}{V} * 100$$

En donde:

- A_S es el acoplamiento de salida del sistema
- A_E es el acoplamiento de entrada del sistema
- V es la cantidad de nodos del sistema
- ψ_S es la cantidad de clases con llamadas de salida que superan el umbral establecido
- ψ_E es la cantidad de clases con llamadas de entrada que superan el umbral establecido

4.3.2.15 Herencia

En el paso b del modelado del comportamiento del sistema mediante redes complejas se asentaron todos los métodos utilizados en la ejecución del sistema agregando un asterisco en aquellos heredados. Además de obtener la información a partir de las salidas de este paso, también es posible identificar los métodos heredados a partir del diagrama de clases. Conociendo los métodos heredados, interesa en esta métrica determinar la proporción de llamadas totales que representan los métodos heredados sobre el total de llamadas del sistema. De forma numérica, estará dada por:

$$H = \frac{\sum_1^E \alpha_E}{LL} * 100$$

Siendo:

- E la cantidad de métodos heredados
- α_E la cantidad de veces que se ejecuta el método heredado E
- LL la cantidad de llamadas totales del sistema

4.3.2.16 Resumen de métricas propuestas en esta tesis

Métrica		Definición
D	*V*	Cantidad de clases del sistema
Cantidad de aristas	*E*	Cantidad de métodos del sistema
Llamadas totales	*LL*	Cantidad de ejecuciones de métodos en el sistema

Promedio de llamadas por método	*LL(M)*	Promedio de llamadas por método
Promedio de llamadas por clase	*LL(E)*	Promedio de llamadas por clase
Métodos sin uso	M_0	Cantidad de métodos no utilizados
Métodos con un único uso	M_1	Cantidad de métodos utilizados en una única oportunidad
Métodos muy utilizados	μ	Métodos con un 50% más de llamadas que el promedio
Tendencia central de la complejidad	*CP* *DPC* *CP'*	Media, desviación estándar y mediana de la complejidad del sistema.
Complejidad dinámica total del sistema	*CDS*	Sumatoria de la complejidad del sistema
Cohesión	*G*	Promedio de cohesiones de la clases de un sistema
Polimorfismo	*P*	Proporción entre llamadas a métodos polimórficos y llamadas totales
Sobrecarga	*S*	Proporción entre llamadas a métodos sobrecargados y llamadas totales
Acoplamiento	A_S A_E	Clases con un 50% más de llamadas de salida o entrada que el promedio de llamadas por clase
Herencia	*H*	Proporción entre llamadas a métodos heredados y llamadas totales

Tabla 4.5 – Resumen de métricas propuestas en esta tesis

4.3.2 Métricas existentes en otros dominios, reinterpretadas al dominio de sistemas

En esta sección se presenta un conjunto de métricas tomadas del dominio de las redes complejas y reinterpretadas al dominio de sistemas.

4.3.2.1 Factor small-world

Tal como se analizó en la sección 2.7.1, la característica small-world determina que en una red compleja la mayoría de los nodos no estarán conectados entre sí pero existirá al menos un camino relativamente corto entre dos nodos cualesquiera. Desde el punto de vista de los sistemas software, es interesante conocer la longitud del camino promedio entre los nodos ya que este valor indicará la amplitud del sistema. Además, la característica small-world es indicativa de la robustez del sistema y de la eficiencia en la comunicación a través del mismo.

La longitud del camino promedio, o factor small-world, no es un valor fácil de obtener y su cálculo está dado por [Albert y Barabási, 2002]:

$$L = \frac{1}{V(V-1)} * \sum_{\substack{1 \\ i \neq j}}^{V} d(v_i, v_j)$$

En donde $d(v_i, v_j)$ es la distancia mínima existente entre los nodos v_i y v_j, la cual debe considerarse cero para los casos en que no haya comunicación entre ambos nodos.

4.3.2.2 Hubbing

Las redes small-world poseen un pequeño conjunto de nodos con un alto grado de distribución, que son conocidos como hubs. Desde el punto de vista dinámico en sistemas, los hubs representan clases con muchas conexiones, es decir, con un uso notablemente mayor al del promedio de clases. Es interesante conocer las clases hub ya que al ser clases muy utilizadas dan lugar a posibilidades de mejora, además de evidenciar aquellas clases cuyo error podría ser catastrófico para el sistema en ejecución.

El concepto de hubbing está íntimamente relacionado con el de acoplamiento, tal como fue visto en la sección 4.3.1.14, el cual será base para el cálculo del hubbing en un sistema. Considerando como ψ_H a las clases con una cantidad de llamadas de entrada y/o salida que supere el umbral establecido $\alpha_\psi = 1{,}5 * LL(V)$, definiremos como hubbing a:

$$\delta = \frac{\psi_H}{V} * 100$$

4.3.2.3 Factor scale-free

Una de las características distintivas de las redes complejas es la distribución del grado de los nodos, o cantidad de aristas que posee. En las redes aleatorias el grado de los nodos tendrá una distribución de Poisson mientras que en las redes complejas varía notablemente, tal como se estudió en la sección 2.7.1.

El modelo de preferencias dinámicas explicado por Barabási [Barabási, 2002] determina que el grado de distribución en una red compleja está dado por una ley de potencias que sigue la forma $P(k) \sim k^{-\gamma}$ en donde normalmente $2 < \gamma < 3$. Esto significa que un pequeño conjunto de nodo tendrá una gran cantidad de aristas (nodos hubs) mientras que la mayoría de los nodos tendrá una cantidad notablemente menor de aristas.

Tal como se estudió en la sección 2.7.2 ya ha quedado demostrado en la literatura de la materia que los sistemas software siguen una ley de potencias de la forma descripta en el párrafo anterior, pero esos estudios quedaron acotados a análisis estáticos. Interesa conocer en esta métrica el grado de distribución en un sistema en ejecución, considerando como grado a la cantidad de llamadas existentes en cada nodo.

A tal fin, esta métrica calcula la distribución de grado dinámica a partir de contabilizar para cada nodo la cantidad llamadas que posee, contabilizando α en todas sus aristas. Posteriormente es necesario clasificar la cantidad de nodos que posee cada grado dinámico y determinar su incidencia con respecto a la cantidad de nodos totales.

Finalmente, se podrán representar los resultados en un gráfico de dos ejes, en donde el eje x representará el grado de distribución y el eje y representará la fracción de nodos que posee dicha distribución. La curva resultante que se aproxime al gráfico determinará la el valor de γ.

4.3.2.4 Coeficiente de agrupamiento

Este concepto, introducido por Watts y Strogatz [Watts y Strogatz, 1998], tiene sus orígenes en la sociología en donde es conocido como fracción de triples transitivos. El coeficiente de agrupamiento representa la tendencia a la formación de cliques, es decir, conjuntos de vértices tal que para todo par de vértices de dicho conjunto existe una arista que las conecta.

Albert y Barabási [Albert y Barabási, 2002] definen al coeficiente de agrupamiento partiendo de un nodo v_i que tiene a su vez k_i aristas que lo conectan a k_i nodos. Si los nodos que son vecinos inmediatos del nodo v_i formaran un clique habría k_i (k_i – 1)/2 aristas entre ellos. Podemos entonces definir como coeficiente de clustering (C_i) a la razón existente entre el número total de aristas (E_i) y la cantidad de aristas (k_i (k_i – 1)/2) que existiría en caso de que formaran un clique. Es decir:

$$C_i = \frac{2\ E_i}{k_i(k_i - 1)}$$

El coeficiente de agrupamiento de toda la red será el promedio de todos los C_i. De forma numérica:

$$C = \frac{1}{i}\sum C_i$$

Este valor representará la dependencia promedio existente entre las clases del sistema y variará entre 0 y 1, siendo mínimo cuando exista un bajo grado de conexión o dependencia entre las clases y máximo cuando todas las clases estén conectadas con todas las demás.

En un grafo aleatorio, en el cual los nodos y sus conexiones se distribuyen aleatoriamente según una probabilidad p, el coeficiente de agrupamiento es C = p, pero en la mayoría, sino todos, los sistemas reales se demuestra que, manteniendo la cantidad de nodos, el coeficiente de agrupamiento es mucho mayor [Albert y Barabási, 2002; Watts y Strogatz, 1998].

Al analizar un sistema es importante considerar que no debería darse para ningún caso que $C_i = 0$ ya que esto implicaría que la clase no tiene ninguna conexión, o bien, que no es utilizada en ningún momento. Tampoco debería darse que C = 1 ya que implicaría que todas las clases invocan a todas las demás clases resultando en un acoplamiento máximo, contrario a las buenas prácticas de diseño.

4.3.2.5 Resumen de métricas existentes en otros dominios, reinterpretadas al dominio de sistemas

Métrica		Definición
Factor small-world	L	Longitud del camino promedio entre clases
Hubbing	δ	Clases con un 50% más de llamadas que el promedio
Factor scale-free	γ	Exponente del grado de distribución de las clases
Coeficiente de agrupamiento	C	Dependencia promedio entre las clases del sistema

Tabla 4.6 – Resumen de métricas existentes en otros dominios, reinterpretadas al dominio de sistemas

5. VALIDACIÓN

En este capítulo se analiza y estudia el comportamiento de la solución propuesta en esta tesis. En primer lugar se realiza una introducción al método de validación a aplicar, con las justificaciones correspondientes de su elección (sección 5.1). Posteriormente se explican las partes de la solución que se someterán a estudio (sección 5.2), presentándose al final los resultados de la aplicación del método de validación (sección 5.3).

5.1 Introducción al método de validación

Visto y considerando que la solución propuesta en el capítulo anterior consiste en una metodología adaptable a diferentes procesos de desarrollo de software y a sistemas de variadas características y tamaños, sumado al hecho de que la comunidad de métricas de software tiene una marcada preferencia por las métricas validadas empíricamente [Kitchembaum, 2010], se determinó utilizar un método de validación empírico por simulación basado en el método de Monte Carlo.

Desarrollados sistemáticamente a mediados de la década de 1940, los métodos de Monte Carlo son definidos formalmente como la acción de representar la solución de un problema como un parámetro de una población hipotética y usar una secuencia numérica aleatoria de números para construir una muestra de la población a partir de la cual estimaciones estadísticas del parámetro pueden ser obtenidas [Halton, 1970]. En otras palabras, los métodos de Monte Carlo son un amplio conjunto de algoritmos que a partir de la ejecución de experimentos sobre muestras aleatorias permiten obtener un resultado numérico [Kalos y Whitlock, 2008].

Los métodos de Monte Carlo dan solución a una variedad de problemas, estocásticos o determinísticos, y fueron favorecidos notablemente por el incremento en la capacidad computacional ya que su precisión está asociada a la cantidad de experimentos ejecutados.

La aplicación de los conceptos de los métodos de Monte Carlo a la simulación consiste en el desarrollo de un modelo matemático del proceso a analizar, identificando en dicho proceso las variables independientes y las variables dependientes, quedando el comportamiento de estas últimas definido por el valor de las primeras. Luego de definido el modelo matemático del proceso a simular, y de la identificación de las variables correspondientes, se debe proceder ejecutando experimentos sobre dicho modelo, asignando valores a las variables independientes y observando el comportamiento asociado de las variables dependientes, las cuales permitirán comprender el funcionamiento del sistema tras una adecuada cantidad de experimentos ejecutados.

La metodología presentada en el capítulo anterior posee dos actividades principales: el modelado del sistema como red compleja y el cálculo de métricas orientadas al diseño. Considerando que la primera

de estas actividades está acotada a la representación de los diagramas de secuencia de UML como una red compleja, la validación se aplicará únicamente sobre la segunda actividad, en la cual se presentaron una serie de métricas para el estudio de la dinámica de los sistemas desde el punto de vista del diseño. A su vez, del conjunto de métricas presentadas, solo se realizará la validación de las métricas propuestas en esta tesis ya que se considera que las métricas existentes en otros dominios ya fueron validadas en los mismos y solamente se reinterpretaron al dominio de sistemas software, al ser los mismos modelados como redes complejas.

5.2 Aplicación del método de validación

En la sección 4.3, *Cálculo de métricas orientadas al diseño de sistemas*, se presentó una serie de métricas que permiten estudiar el diseño de un sistema desde un punto de vista dinámico. Estas métricas pueden ser clasificadas según el estándar ISO/IEC 9126 en métricas básicas (o directas), métricas de agregación y métricas derivadas, tal como se describió en el estado de la cuestión.

Las métricas básicas, que se obtienen sin ningún cálculo directamente del modelo del sistema, se corresponden dentro de los métodos de Monte Carlo con las variables independientes y son las que definen las características intrínsecas del sistema en estudio. Las métricas de agregación y las métricas derivadas, las cuales requieren de cálculos para su obtención, se corresponden a su vez con las variables dependientes ya que su valor dependerá funcionalmente de las métricas básicas. Solamente se realizará la validación sobre las métricas derivadas ya que las métricas básicas y las métricas de agregación presentan características propias del sistema en estudio por lo que no se requiere estudiar su comportamiento.

La validación de las métricas desarrolladas en esta tesis mediante una simulación basada en el método de Monte Carlo partirá de un conjunto de sistemas de prueba generados de forma aleatoria de acuerdo a una distribución de probabilidad delimitada por los atributos característicos de sistemas definidos por Myers [Myers, 2003]. Los límites serán establecidos a partir de una serie de parámetros experimentales y acotarán a ciertos atributos de los sistemas de prueba, los cuales dentro del proceso de validación se corresponden con las variables independientes, o métricas básicas. A partir de las métricas básicas, a su vez, se calcularán las demás métricas realizando las interpretaciones correspondientes para cada una de ellas. A continuación se describe el protocolo definido para el método de validación:

1. La primera actividad consiste en la definición de una serie de parámetros para todas las variables independientes del modelo de simulación. Se deberá definir un límite inferior y/o un límite superior para los siguientes atributos de los sistemas del banco de pruebas: cantidad de clases, cantidad de métodos por clase, cantidad de llamadas asociada a cada método,

complejidad de cada método, umbral de cohesión, umbral de polimorfismo y sobrecarga y umbral de herencia.

2. Una vez definidos los parámetros iniciales se deberá generar un banco de pruebas de una cantidad determinada de sistemas. Se utilizará la aplicación Microsoft Excel para registrar los parámetros definidos en la primera actividad y todos los datos correspondientes a los sistemas del banco de pruebas, utilizando la herramienta de datos Análisis de Hipótesis de dicha aplicación.
3. A partir de las simulaciones realizadas sobre todos los sistemas del banco de pruebas, se realizará una integración estadística de los resultados generando los gráficos y tablas de soporte que se consideren necesarias
4. Finalmente, se interpretarán los resultados obtenidos elaborando guías de interpretación para las métricas desarrolladas

5.3 Estudio de las métricas de MADDS

En esta sección, y en base al protocolo definido para el método de validación descripto en la sección anterior, se especifican las variables independientes (sección 5.3.1), las variables dependientes (sección 5.3.2) y se analizan los resultados del proceso de validación de cada una de las variables desarrolladas en esta tesis (sección 5.3.3).

5.3.1 Variables independientes

En la primera actividad del protocolo definido para el método de validación se enumeraron los atributos de los sistemas que se corresponden con las variables independientes. Estas variables serán generadas aleatoriamente dentro del proceso de simulación y se encontrarán acotadas por parámetros que permitirán modelar sistemas de acuerdo a lo descripto por los atributos característicos de sistemas modelados como redes complejas definidos por Myers [Myers, 2003].

Además de afectar directamente a las métricas de agregación y a las métricas derivadas, una de las variables independientes (Cantidad de nodos) es a su vez una métrica básica ya que se obtiene directamente del sistema en estudio sin necesidad de ningún cálculo o uso de otra métrica. En la tabla 5.1 se enumeran las variables independientes junto con una descripción de las mismas.

Variable independiente	Descripción
Cantidad de nodos (V)	Cantidad de clases del sistema. Se corresponde con la métrica cantidad de nodos (*V*). Se le asignará un valor aleatorio dentro de un rango de valores definido a nivel experimental.
Cantidad de métodos por clase (E_V)	Cantidad de métodos por cada clase del sistema, o cantidad de aristas por cada nodo. Se le asignará un valor aleatorio dentro de un rango de valores definido a nivel experimental.

Cantidad de llamadas de cada método (α)	Es la cantidad de veces que cada método es ejecutado en la simulación de la corrida del programa. Se le asignará un valor aleatorio dentro de un rango de valores definido a nivel experimental.
Complejidad de cada método (C_E)	Complejidad ciclomática [McCabe, 1976] de cada método. Se le asignará un valor aleatorio dentro de un rango de valores definido a nivel experimental.
Umbral de cohesión (U_c)	Porcentaje de métodos cohesivos. Se le asignará un valor aleatorio dentro de un rango de valores definido a nivel experimental.
Umbral de polimorfismo y sobrecarga (U_p)	Porcentaje de métodos con polimorfismo y sobrecarga. Se le asignará un valor aleatorio dentro de un rango de valores definido a nivel experimental.
Umbral de herencia (U_h)	Porcentaje de métodos heredados. Se le asignará un valor aleatorio dentro de un rango de valores definido a nivel experimental.

Tabla 5.1 – Listado de variables independientes

5.3.2 Variables dependientes

Dentro del modelo definido para el método de validación, las variables dependientes son las métricas de agregación y las métricas derivadas propuestas en esta tesis. Las métricas de agregación son aquellas que se obtienen a través de cálculos directos sobre las métricas básicas y son: la métrica de cantidad de aristas, la métrica de llamadas totales, la métrica de cantidad de métodos sin uso y la métrica de cantidad de métodos con un único uso. Tal como se mencionó anteriormente, las métricas derivadas no requieren del estudio de su comportamiento. A su vez, las métricas derivadas, que requieren el uso de más de una métrica básica o la aplicación de alguna función sobre las mismas, son: la métrica del promedio de llamadas por método, la métrica del promedio de llamadas por clase, la métrica de métodos muy utilizados, la métrica de tendencia central de la complejidad, la métrica de complejidad dinámica total del sistema, la métrica de cohesión, la métrica de polimorfismo, la métrica de acoplamiento y la métrica de herencia.

En la tabla 5.2 se enumeran las variables dependientes y se indican las demás variables, dependientes o independientes, de las cuales depende. Es importante considerar que algunas métricas, como la tendencia central de la complejidad, poseen más de un cálculo.

Variable dependiente	Dependencia de otras variables
Cantidad de aristas	Depende de la cantidad de métodos por clase
Llamadas totales	Depende de la cantidad de llamadas por método
Métodos sin uso	Depende de la cantidad de llamadas por método
Métodos con un único uso	Depende de la cantidad de llamadas por método
Promedio de llamadas por método	Depende de las llamadas totales y de la cantidad de aristas
Promedio de llamadas por clase	Depende de las llamadas totales y de la cantidad de clases
Métodos muy utilizados	Depende del promedio de llamadas por método y de la cantidad de aristas

Tendencia central de la complejidad	Depende de la complejidad de cada método, de la cantidad de llamadas por método, de la cantidad de aristas y de la cantidad de clases
Complejidad dinámica total	Depende de la complejidad de cada método y de la cantidad de llamadas por método
Cohesión	Depende del umbral de cohesión y de la cantidad de llamadas por método
Polimorfismo	Depende del umbral de polimorfismo y sobrecarga y de la cantidad de llamadas por método
Sobrecarga	Depende del umbral de polimorfismo y sobrecarga y de la cantidad de llamadas por método
Acoplamiento	Depende del promedio de llamadas por clase y de la cantidad de clases
Herencia	Depende de las llamadas totales y de la cantidad de llamadas por método

Tabla 5.2 – Listado de variables dependientes

5.3.3 Validación de métricas desarrolladas en esta tesis

En esta sección se realiza el análisis de las métricas derivadas desarrolladas en esta tesis, aplicando el protocolo definido para el método de validación y con el objetivo de obtener una regla experimental del funcionamiento de las mismas. Se analizan, así la métrica del promedio de llamadas por método (sección 5.3.3.1), la métrica del promedio de llamadas por clase (sección 5.3.3.2), la métrica de métodos muy utilizados (sección 5.3.3.3), la métrica de tendencia central de la complejidad (sección 5.3.3.4), la métrica de complejidad dinámica total del sistema (sección 5.3.3.5), la métrica de cohesión (sección 5.3.3.6), las métricas de polimorfismo y sobrecarga (sección 5.3.3.7), la métrica de acoplamiento (sección 5.3.3.8) y la métrica de herencia (sección 5.3.3.9).

5.3.3.1 Métrica del promedio de llamadas por método

El estudio de la métrica del promedio de llamadas por método requiere el uso de las variables experimentales que se describen en la tabla 5.3.

Variable experimental	Descripción
LL(M)	Promedio de llamadas por método. Se define mediante la fórmula $LL(M) = LL \,/\, E$
LL	Llamadas totales. Es la cantidad total de invocaciones o llamadas a métodos que se realizan durante la ejecución del sistema. Se define como $LL = \sum \alpha$. Es una métrica de agregación.
α	Cantidad de llamadas por método
E	Cantidad de aristas o métodos del sistema
V	Cantidad de nodos. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.3 – Variables experimentales de la métrica del promedio de llamadas por método

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.4, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.4 – Valores definidos para las variables experimentales de la métrica del promedio de llamadas por método

Del análisis de los datos generados por los experimentos ejecutados se observa que la métrica del promedio de llamadas por método efectivamente mide la cantidad de veces que, en promedio, cada método es llamado a lo largo de la ejecución del sistema. La modificación de los parámetros en el experimento permite observar que esta métrica es directamente proporcional, de forma lineal, a la cantidad de llamadas e inversamente proporcional, de forma lineal, a la cantidad de métodos. Para los valores aleatorios asignados a las variables independientes tanto el promedio como la mediana de esta métrica se mantienen en aproximadamente 21 llamadas por método, aún al considerar un banco de pruebas de 10.000 sistemas, incluyendo sistemas de entre 51 y 627 métodos y de entre 1233 y 11717 llamadas a métodos.

El conocer la carga promedio de un método en la ejecución de un sistema permite hacer una valoración de la importancia de cada método en el sistema y permite realizar comparaciones entre sistemas de diferentes características y tamaños. Como regla experimental del comportamiento de esta métrica, se concluye que la misma aumentará linealmente junto con la cantidad de llamadas totales y disminuirá linealmente junto con la cantidad de métodos totales del sistema.

5.3.3.2 Métrica del promedio de llamadas por clase

La métrica del promedio de llamadas por clase tiene propiedades similares a la métrica del promedio de llamadas por método aunque se encuentra orientada a medir la carga promedio que tienen las clases en el sistema. El estudio de la métrica del promedio de llamadas por clase requiere el uso de las variables experimentales que se describen en la tabla 5.5.

Variable experimental	Descripción
LL(V)	Promedio de llamadas por clase. Se define mediante la fórmula $LL(V) = LL \,/\, V$
LL	Llamadas totales. Es la cantidad total de invocaciones o llamadas a métodos que se realizan durante la ejecución del sistema. Se define como $LL = \sum \alpha$. Es una métrica de agregación.
α	Cantidad de llamadas por método
E	Cantidad de aristas o métodos del sistema. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
V	Cantidad de nodos
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.5 – Variables experimentales de la métrica del promedio de llamadas por clase

Para el análisis del comportamiento de esta métrica se define un diseño experimental similar al generado para el análisis de la métrica del promedio de llamadas por método, consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. En la tabla 5.6 se establecen los valores definidos para las variables independientes involucradas en esta métrica, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.6 – Valores definidos para las variables experimentales de la métrica del promedio de llamadas por clase

Los experimentos simulados permiten llegar a la conclusión de que efectivamente la métrica del promedio de llamadas por clase mide la cantidad de veces que, en promedio, cada clase recibe una invocación a un método. Al igual que en la métrica del promedio de llamadas por método, esta métrica es directamente proporcional a la cantidad de llamadas e inversamente proporcional a la cantidad de clases, siendo ambas proporciones lineales. En los experimentos realizados, y con los valores asignados aleatoriamente dentro de los parámetros definidos, se obtuvo un promedio de 225 llamadas por clase, con una mediana similar y un mínimo de 138 llamadas promedio y un máximo de 288. Valores similares se obtuvieron ampliando el universo de experimentos a 10.000 sistemas.

Como regla experimental del comportamiento de esta métrica, se puede afirmar que el valor que obtenga aumentará linealmente con la cantidad de llamadas totales del sistema y disminuirá linealmente junto con la cantidad de clases totales del sistema, desde luego manteniendo constantes

las demás variables del sistema. Esta métrica permite realizar un análisis de la carga promedio de cada clase en la ejecución de un sistema, pudiendo realizar una valoración a partir de la misma.

5.3.3.3 Métrica de métodos muy utilizados

La simulación de la métrica de métodos muy utilizados requiere el uso de las variables experimentales que se describen en la tabla 5.7. Esta métrica parte de la definición de un umbral, correspondiente a una vez y media del promedio de llamadas por método. A partir de dicho umbral se calcula el porcentaje de métodos muy utilizados.

Variable experimental	Descripción
μ	Porcentaje de métodos muy utilizados. Se define como $\mu = (M_\mu / \alpha_\mu) * 100$
M_μ	Cantidad de métodos cuya cantidad de llamadas supera $\alpha\mu$
α_μ	Umbral de métodos muy utilizados. Se define como $\alpha_\mu = 1{,}5 * LL(M)$
LL(M)	Promedio de llamadas por método. Se define mediante la fórmula $LL(M) = LL / E$
LL	Llamadas totales. Es la cantidad total de invocaciones o llamadas a métodos que se realizan durante la ejecución del sistema. Se define como $LL = \sum \alpha$. Es una métrica de agregación.
α	Cantidad de llamadas por método
E	Cantidad de aristas o métodos del sistema
V	Cantidad de nodos. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.7 – Variables experimentales de la métrica de métodos muy utilizados

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.8, obteniendo un banco de pruebas de sistemas de diversos tamaños y características. En las simulaciones realizadas, los sistemas variaron entre los 33 métodos y los 564, con un promedio de 250 y una mediana de 240. Las llamadas a su vez variaron entre las 742 y las 11218, con un promedio de 5459 y una mediana de 5138.

Variable experimental	Valores definidos
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.8 – Valores definidos para las variables experimentales de la métrica de métodos muy utilizados

A partir del análisis experimental se observa que esta métrica toma un valor promedio de aproximadamente 30% con una muy baja dispersión, valor que representa la cantidad de métodos con una cantidad de llamadas superior a 1,5 veces el promedio de llamadas por método. Este valor se condice con las características de los sistemas software modelados como redes complejas, definidas por Myers [Myers, 2003] y expuestas en el capítulo 2, *estado del arte*. Se modificaron los parámetros de entrada del experimento sin encontrar ningún cambio en los resultados al variar los parámetros de cantidad de clases o cantidad de llamadas por método, pero encontrando un cambio notable al modificar la variable de cantidad de métodos por clase. Esta modificación, que se observa en la figura 5.1 permite asumir que existe una cantidad de métodos por clase que maximizará la proporción de métodos muy utilizados.

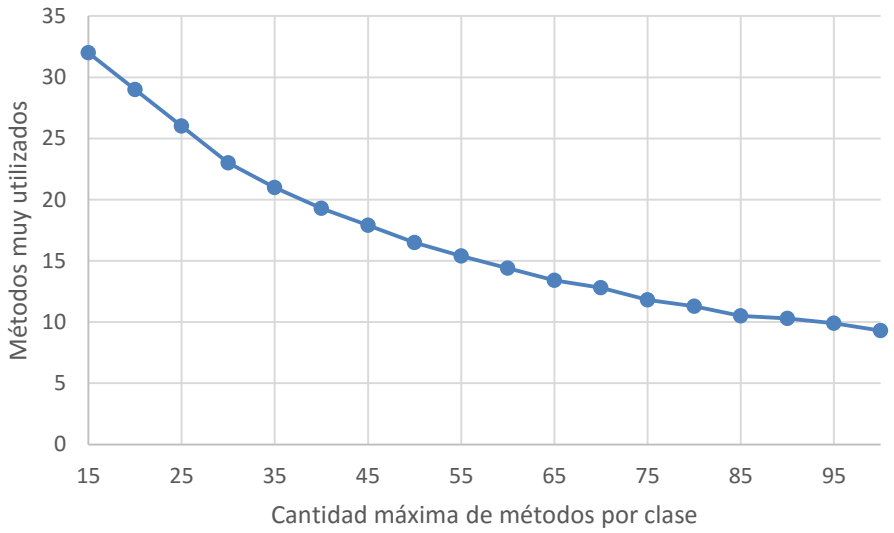


Figura 5.1 – Comportamiento de la métrica de métodos muy utilizados al modificar el parámetro E_v

Conocer el porcentaje de métodos muy utilizados -a partir de la información obtenida de un análisis dinámico de un sistema- permite determinar aquellos métodos con un gran impacto en la ejecución del sistema. El porcentaje de métodos muy utilizados influenciará de forma directa sobre el acoplamiento y de forma inversa sobre la cohesión del sistema por lo que es importante encontrar un punto de equilibrio, al igual que en las demás características de diseño del sistema.

5.3.3.4 Métrica de tendencia central de la complejidad

La medición de la complejidad en esta tesis se desarrolló a través de cinco métricas diferentes, cuatro de las cuales están orientadas a la medición de la tendencia central de la misma. Estas métricas son: la complejidad dinámica promedio (CP), la desviación estándar de la complejidad dinámica promedio (DCP) y la mediana de la complejidad dinámica (MC), las tres operando a nivel de métodos; y la complejidad dinámica promedio por clase (CP') que opera a nivel de clases. En la tabla 5.9 se describen las variables experimentales necesarias para la medición de estas tres métricas.

Variable experimental	Descripción
CP	Complejidad dinámica promedio. Se define como: $CP = \frac{\sum_1^E (C_E * \alpha_E)}{E}$
DCP	Desviación estándar de la complejidad dinámica de los métodos con respecto a la complejidad dinámica promedio (CP). Se define como: $DCP = \sqrt{\frac{1}{E}\sum_1^E ((C_E * \alpha_E) - CP)^2}$
CP'	Complejidad dinámica promedio por clase. Se define como: $CP' = \frac{\sum_1^E (C_E * \alpha_E)}{V}$
MC	Mediana de la complejidad dinámica. Se obtiene mediante el cálculo: $\begin{cases} MC = m_{(n+1)/2} \text{ para un } n \text{ impar} \\ MC = (m_{n/2} + m_{(n/2)+1})/2 \text{ para un } n \text{ par} \end{cases}$
C_E	Complejidad ciclomática [McCabe, 1976] de cada método.
α	Cantidad de llamadas por método
E	Cantidad de aristas o métodos del sistema
V	Cantidad de nodos
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.9 – Variables experimentales de la métrica de tendencia central de la complejidad

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.10, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
C_E	La complejidad ciclomática tomará un valor aleatorio entre 1 y 15, reflejando métodos con un riesgo bajo a moderado [Heitlager *et al*., 2007].
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.10 – Valores definidos para las variables experimentales de la métrica de tendencia central de la complejidad

Luego de analizar los resultados de los diversos experimentos se llega a la conclusión de que para los valores definidos para las variables experimentales todas las mediciones de complejidad toman un valor estable. Considerando la valuación de la complejidad ciclomática de McCabe, la complejidad promedio toma un valor medio de 173 con una desviación estándar de 340 y una mediana de 305, mientras que la complejidad promedio a nivel de clases es de 1806, en promedio. Considerando los

valores obtenidos en los experimentos de las métricas de promedio de llamadas por método y promedio de llamadas por clase, obtenemos que, anulando el factor α, o cantidad de llamadas, la complejidad promedio de cada método es apenas mayor a 8, similar al promedio de las complejidades efectivamente obtenido en los experimentos. Como regla experimental podemos afirmar que todas las métricas de la tendencia central de la complejidad son directamente proporcionales a la complejidad y a la cantidad de llamadas de los métodos e inversamente proporcional a la cantidad de los mismos. Esta métrica, al igual que las anteriores, permite comparar sistemas de distintas características y tamaños entre sí.

5.3.3.5 Métrica de complejidad dinámica total del sistema

El estudio de la métrica de complejidad dinámica total del sistema requiere el uso de las variables experimentales que se describen en la tabla 5.11.

Variable experimental	Descripción
CDS	Complejidad dinámica total del sistema. Se define como: $CDS = \sum_{1}^{E} (C_E * \alpha_E)$
C_E	Complejidad ciclomática [McCabe, 1976] de cada método.
α	Cantidad de llamadas por método
E	Cantidad de aristas o métodos del sistema. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental
V	Cantidad de nodos. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.11 – Variables experimentales de la métrica de complejidad dinámica total del sistema

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.12, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
C_E	La complejidad ciclomática tomará un valor aleatorio entre 1 y 15, reflejando métodos con un riesgo bajo a moderado [Heitlager *et al.*, 2007].
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.12 – Valores definidos para las variables experimentales de la métrica del promedio de llamadas por método

Los experimentos realizados permitan observar que en un universo de sistemas con una complejidad de entre 8488 y 92820, valuada de acuerdo a la métrica de complejidad ciclomática definida por McCabe el promedio es de entre 46000 y 52000, aun ampliando el banco de pruebas a los 10.000 sistemas simulados.

Como regla experimental podemos afirmar que la complejidad dinámica total del sistema es directamente proporcional a la complejidad de los métodos y la cantidad de ejecuciones de los mismos.

5.3.3.6 Métrica de cohesión

El análisis de la métrica de cohesión requiere el uso de las variables experimentales que se describen en la tabla 5.13.

Variable experimental	Descripción
G_C	Cohesión del sistema. Se define numéricamente como: $G_C = \frac{\sum_1^E \alpha_{ENC}}{\sum_1^E \alpha_E}$
U_C	Umbral de cohesión. Determina la frontera sobre la cual un número aleatorio determina si un método es cohesivo o no.
LL	Llamadas totales. Es la cantidad total de invocaciones o llamadas a métodos que se realizan durante la ejecución del sistema. Se define como $LL = \sum \alpha$ o $\sum_1^E \alpha_E$. Es una métrica de agregación.
α	Cantidad de llamadas por método
E	Cantidad de aristas o métodos del sistema. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental
V	Cantidad de nodos. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.13 – Variables experimentales de la métrica de cohesión

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.14, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
U_C	Se le asigna un valor de 0,1 y define que aquellos números aleatorios (entre 0 y 1) menores a dicho umbral sean no cohesivos.
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo

E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.14 – Valores definidos para las variables experimentales de la métrica de cohesión

En base a los resultados obtenidos del experimento se llega a la conclusión de que la métrica de cohesión mide correctamente la proporción entre métodos no cohesivos y métodos cohesivos desde un punto de vista dinámico ya que considera la cantidad de veces que los mismos son ejecutados. En la figura 5.2 se describe gráficamente el comportamiento de la métrica en función del umbral de cohesión, el cual determina la probabilidad de que un método sea cohesivo. Al medir la métrica la proporción con respecto a la totalidad de llamadas a métodos, es correcto tener una relación lineal entre ambas.

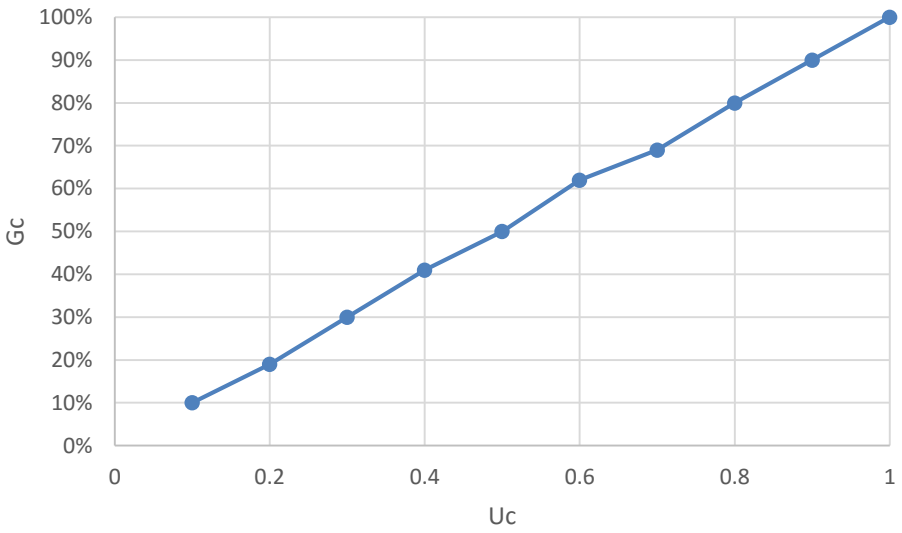


Figura 5.2 – Comportamiento de la métrica de cohesión al modificar el umbral Uc

Como regla experimental, podemos afirmar que la métrica de cohesión es directamente proporcional a la cantidad de veces que se ejecutan los métodos no cohesivos, e inversamente proporcional a la cantidad de llamadas totales a métodos en la ejecución del sistema. Conocer este valor permite no solo comparar sistemas entre sí sino saber qué proporción de métodos no cohesivos hay en el sistema, lo que podría llevar a realizar ajustes en el diseño del sistema.

5.3.3.7 Métricas de polimorfismo y sobrecarga

Si bien el polimorfismo y la sobrecarga son dos aspectos de los sistemas que deben ser evaluados por separado, ambas están orientadas a medir la funcionalidad de los métodos en el sistema. Más allá de esto, ambas métricas son similares entre sí por lo que serán estudiadas en conjunto, a partir de un único umbral de polimorfismo y sobrecarga. El análisis correspondiente a estas métricas, y la definición del umbral compartido, se describe en la tabla 5.15.

Variable experimental	Descripción
P	Polimorfismo del sistema, el cual se define como: $P = \frac{\sum_1^E \alpha_{EP}}{\sum_1^E \alpha_E}$
S	Sobrecarga del sistema, el cual se define como: $S = \frac{\sum_1^E \alpha_{ES}}{\sum_1^E \alpha_E}$
U_p	Umbral de polimorfismo y sobrecarga. Determina la frontera sobre la cual un número aleatorio determina si un método es polimórfico y/o se encuentra sobrecargado
LL	Llamadas totales. Es la cantidad total de invocaciones o llamadas a métodos que se realizan durante la ejecución del sistema. Se define como $LL = \sum \alpha$ o $\sum_1^E \alpha_E$. Es una métrica de agregación.
α	Cantidad de llamadas por método
E	Cantidad de aristas o métodos del sistema. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental
V	Cantidad de nodos. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.15 – Variables experimentales de las métricas de polimorfismo y sobrecarga

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.16, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
U_P	Se le asigna un valor de 0,5. Define que aquellos números aleatorios (entre 0 y 1) menores a dicho umbral sean no cohesivos.
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.16 – Valores definidos para las variables experimentales de las métricas de polimorfismo y sobrecarga

Los resultados del experimento permiten llegar a conclusiones similares a las de la métrica de cohesión vista en la sección anterior. La métrica de polimorfismo mide correctamente la proporción entre métodos polimórficos y métodos totales desde un punto de vista de la ejecución; mientras que la métrica de sobrecarga mide la correcta proporción entre métodos sobrecargados y métodos totales desde un punto de vista dinámico. Al compartir en el experimento el mismo umbral, ambas métricas llegan a valores similares a lo largo de todo el experimento. Además de esto, al partir de una

caracterización aleatoria de los métodos para determinar su polimorfismo o sobrecarga, se llega al mismo fenómeno de correspondencia lineal entre el umbral correspondiente y la métrica en estudio.

Como regla experimental, podemos afirmar que la métrica de polimorfismo es directamente proporcional a la cantidad de veces que se ejecutan los métodos polimórficos, e inversamente proporcional a la cantidad de llamadas totales a métodos en la ejecución del sistema. Misma conclusión puede hacerse con respecto a la métrica de métodos sobrecargados. Estas métricas permiten comparar diferentes sistemas entre sí, además de proporcionar guías para evaluar el diseño del sistema.

5.3.3.8 Métrica de acoplamiento

El análisis de la métrica de acoplamiento requiere el uso de las variables experimentales que se describen en la tabla 5.17. De forma análoga a la métrica de métodos muy utilizados, la métrica de acoplamiento parte de la definición de un umbral, correspondiente a una vez y media el promedio de llamadas por clase, a partir del cual se establecen aquellos métodos con un elevado nivel de acoplamiento de entrada o de salida.

Variable experimental	Descripción
A_S	Porcentaje de clases con elevado acoplamiento de salida. Se define numéricamente como: $A_S = (\psi_S / V) * 100$
A_E	Porcentaje de clases con elevado acoplamiento de entrada. Se define numéricamente como: $A_E = (\psi_E / V) * 100$
ψ_S	Cantidad de clases que realizan más de α_ψ llamadas
ψ_E	Cantidad de clases que reciben más de α_ψ llamadas
α_ψ	Umbral de acoplamiento. Se define como = 1,5 * LL(V)
LL(V)	Promedio de llamadas por clase. Se define mediante la fórmula $LL(V) = LL / V$
LL	Llamadas totales. Es la cantidad total de invocaciones o llamadas a métodos que se realizan durante la ejecución del sistema. Se define como $LL = \sum \alpha$. Es una métrica de agregación.
α	Cantidad de llamadas por método. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental
V	Cantidad de nodos. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.17 – Variables experimentales de la métrica de acoplamiento

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.18, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.18 – Valores definidos para las variables experimentales de la métrica de acoplamiento

Se realizaron dentro del experimento decenas de simulaciones encontrando una dispersión de los resultados en función de las características del sistema. En la mayoría de las ejecuciones del experimento el acoplamiento fue relativamente bajo, cercano al 5%, y nunca superó el 20%. Este resultado se corresponde con las buenas prácticas de diseño de sistemas, en las cuales se sugiere un bajo acoplamiento. Asimismo, permite comprobar las características de los sistemas software modelados como redes complejas, definidas por Myers [Myers, 2003]. Conocer el porcentaje de clases con un alto acoplamiento de salida o de entrada, permite establecer el porcentaje de clases con un muy alto impacto en la ejecución del sistema ya que este parámetro se establece en función de la cantidad de llamadas totales de las clases, tanto para la salida como para la entrada. Esta métrica permite además comparar sistemas entre sí.

5.3.3.9 Métrica de herencia

El análisis de la métrica de herencia requiere el uso de las variables experimentales que se describen en la tabla 5.19.

Variable experimental	Descripción
H	Herencia dinámica del sistema. Se define numéricamente como: $\mathrm{H} = \frac{\sum_1^E \alpha_E}{LL} * 100$
U_H	Umbral de Herencia. Determina la frontera sobre la cual un número aleatorio determina si un método es heredado o no.
LL	Llamadas totales. Es la cantidad total de invocaciones o llamadas a métodos que se realizan durante la ejecución del sistema. Se define como $LL = \sum \alpha$ o $\sum_1^E \alpha_E$. Es una métrica de agregación.
α_E	Cantidad de llamadas por método heredado
E	Cantidad de aristas o métodos del sistema. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental
V	Cantidad de nodos. Si bien no tiene influencia directa en el cálculo de esta métrica, es requerida para el diseño experimental
E_V	Cantidad de métodos por nodo. No tiene influencia directa en el cálculo de esta métrica pero es requerida para el diseño experimental

Tabla 5.19 – Variables experimentales de la métrica de herencia

Para analizar el comportamiento de esta métrica se define un diseño experimental consistente en un banco de pruebas de 1.000 sistemas simulados, de acuerdo a los requerimientos planteados en la

sección 5.2, *aplicación del método de validación*. A tal fin, para cada variable independiente se establece un valor aleatorio acotado por los parámetros experimentales definidos en la tabla 5.20, obteniendo un banco de pruebas de sistemas de diversos tamaños y características.

Variable experimental	Valores definidos
U_C	Se le asigna un valor de 0,6 y define que aquellos números aleatorios (entre 0 y 1) menores a dicho umbral sean heredados.
α	La cantidad de llamadas por método tendrá un valor aleatorio de entre 0 y 100 llamadas a lo largo de la simulación del experimento
V	La cantidad de clases tomará un valor aleatorio para cada sistema entre 5 y 50 clases, considerando que MADDS permite modelar desde una parte de un sistema hasta un sistema completo
E_V	La cantidad de métodos por clase tendrá un valor aleatorio de entre 1 y 20, considerando que MADDS analiza a cada llamada sobrecargada o polimórfica de un método como un método diferente

Tabla 5.20 – Valores definidos para las variables experimentales de la métrica de herencia

Los resultados del experimento permiten llegar a la conclusión de que la métrica de herencia mide correctamente la proporción de llamadas totales que representan los métodos heredados sobre el total de llamadas del sistema, siempre desde el punto de vista dinámico. En los experimentos realizados, la métrica de herencia arrojó una media aritmética de 61%, similar a la mediana, con un mínimo de 49% y un máximo de 79%

Como regla experimental, podemos afirmar que la métrica de herencia es directamente proporcional a la cantidad de veces que se ejecutan los métodos heredados, e inversamente proporcional a la cantidad de llamadas totales del sistema. Conocer este valor permite no solo comparar sistemas entre sí sino saber qué proporción de métodos heredados hay en el sistema, lo que podría llevar a realizar ajustes en el diseño del sistema.

6. CONCLUSIONES

En este capítulo se presentan los aportes de esta tesis de maestría (sección 6.1) y se identifican futuras líneas de investigación en base al problema abierto sobre el cual se trabajó (sección 6.2).

6.1 Aportes de esta tesis

En esta tesis se presentó una metodología de análisis dinámico de sistemas basada en redes complejas, la cual pretende dar soporte a la toma de decisiones, tanto técnicas como de gestión, del proceso de desarrollo de software. La metodología desarrollada propone una serie de métricas que permiten evaluar el comportamiento del sistema en la etapa de diseño de sistemas, cubriendo así dos aspectos descuidados en las métricas desarrolladas hasta el momento: la precisión y la oportunidad. Precisión, al utilizar métricas dinámicas y oportunidad, al desarrollar las mismas en una la fase de diseño de sistemas, fase temprana del proceso de desarrollo de software.

De forma más específica, se listan a continuación los aportes de la metodología de análisis desarrollada al cuerpo de conocimiento de la Ingeniería del Software:

I. El desarrollo de una metodología de análisis dinámico de sistemas basada en redes complejas cuyos componentes pueden ser reemplazados, o bien, a la que se pueden agregar nuevos componentes.
II. La capacidad de modelar la ejecución de un sistema como una red compleja, aportando una nueva visión a los avances realizados en el modelado de sistemas software como redes complejas, los cuales se encontraban acotados hasta el momento a aspectos estructurales de los sistemas.
III. La propuesta de una serie de métricas, desarrolladas en esta tesis, que permiten evaluar el comportamiento en ejecución de un sistema. Se han propuesto las métricas de cantidad de nodos, cantidad de aristas, llamadas totales, promedio de llamadas por método, promedio de llamadas por clase, métodos sin uso, métodos con un único uso, métodos muy utilizados, tendencia central de la complejidad, complejidad dinámica total, cohesión, polimorfismo, sobrecarga, acoplamiento y herencia.
IV. El uso de métricas y conceptos del universo de las redes complejas, reinterpretados al dominio de sistemas. Se desarrollaron así las métricas de factor small-world, hubbing, factor free-scale y coeficiente de agrupamiento.

Los aportes realizados permiten responder a las preguntas de investigación planteadas en el capítulo 3, las cuales fueron la guía de desarrollo de esta obra y que se responden a continuación:

Pregunta 1: ¿Es posible desarrollar un modelo de sistemas software basado en redes complejas, generable a partir de diagramas UML, basado en RUP, centrado en el diseño, y con el objetivo de calcular métricas sobre el mismo?

Respuesta 1: Se pudo desarrollar un modelo de sistemas software basado en redes complejas, generado a partir de diagramas UML, basado en RUP, centrado en el diseño y a partir del cual se calculan métricas. El modelo, que extiende los conceptos de las redes complejas para incluir datos necesarios para el análisis de la ejecución de los sistemas, es generado a partir de diagramas de secuencia (UML). Asimismo, la metodología dentro de la cual se desarrolla el modelo utiliza terminología RUP y se encuadra dentro del flujo de análisis y diseño del mismo, flujo en el cual se desarrolla el diseño del sistema.

Pregunta 2: De ser posible desarrollar este modelo, ¿se puede generar un conjunto de métricas que estudien el comportamiento del sistema? En caso afirmativo, ¿qué métricas?

Respuesta 2: Efectivamente a partir del modelo desarrollado se pudo generar un conjunto de métricas para analizar y estudiar el comportamiento de un sistema en ejecución. A tal fin se desarrolló una serie de métricas, además de haber reinterpretado métricas ya existentes del dominio de las redes complejas al dominio de sistemas. Las métricas desarrolladas son las métricas de cantidad de nodos, cantidad de aristas, llamadas totales, promedio de llamadas por método, promedio de llamadas por clase, métodos sin uso, métodos con un único uso, métodos muy utilizados, tendencia central de la complejidad, complejidad dinámica total, cohesión, polimorfismo, sobrecarga, acoplamiento y herencia. Las métricas reinterpretadas son las métricas de factor small-world, hubbing, factor free-scale y coeficiente de agrupamiento

6.2 Futuras líneas de investigación

Durante el desarrollo de esta tesis se identificaron problemas abiertos dentro del área de las métricas de software. A partir de estos problemas identificados se proponen las siguientes líneas de investigación:

I. El estudio del comportamiento de las métricas desarrolladas en esta tesis fue realizado mediante un proceso de validación empírico por simulación basado en el método de Monte Carlo. Surge entonces la posibilidad de evaluar el comportamiento de las mismas en el desarrollo de sistemas reales.

II. Al haberse planteado en esta tesis una metodología de análisis dinámico de sistemas basada en redes complejas cuyos componentes pueden ser reemplazados, surgen dos posibilidades de desarrollo:

   a. Si bien en la actualidad UML es el lenguaje de modelado estándar dentro de la Ingeniería del Software y los diagramas de secuencia son comunes a la mayoría de los desarrollos, podrían desarrollarse nuevas actividades para modelar el comportamiento de un sistema a partir de otros modelos de UML, como los diagramas de comportamiento, o directamente a partir de otros lenguajes de modelado
   b. El proceso unificado de desarrollo de software es el proceso más elegido a la hora de avanzando con nuevos desarrollos de software. Aun así, un gran porcentaje de los desarrollos son realizados bajo otras metodologías, en particular, metodologías ágiles. Es factible incorporar dentro de la metodología desarrollada en esta tesis los componentes necesarios para adaptarla a los procesos de desarrollo ágiles.

III. Se definió un conjunto de métricas para evaluar el comportamiento de la dinámica de un sistema desde el punto de vista de diseño. Es posible desarrollar nuevas métricas sobre la metodología propuesta para cubrir aspectos del diseño que no hayan sido tenidos en cuenta.

IV. El análisis dinámico planteado fue acotado a la cantidad de ejecuciones de cada método y podría ser ampliado mediante la incorporación del tiempo, estimado o real, de ejecución de cada método. Esta incorporación permitiría aumentar aún más la precisión de las métricas desarrolladas.

V. El desarrollo de MAADS tuvo en cuenta la oportunidad de la información, considerando el análisis a partir del diseño del sistema. Sería factible adaptar el estudio que realiza MAADS a partir de código completamente desarrollado.

7. REFERENCIAS

Abreu, F., & Melo, W. (1996, March). Evaluating the impact of object-oriented design on software quality. In Software Metrics Symposium, 1996. Proceedings of the 3rd International (pp. 90-99). IEEE.

Albert, R., & Barabási, A. L. (2002). Statistical mechanics of complex networks. Reviews of modern physics, 74(1), 47.

ANSI/IEEE (2007). "Draft IEEE Standard for software and system test documentation". ANSI/IEEE Std. P829-2007.

Archer, C., & Stinson, M. (1995). Object-Oriented Software Measures (No. CMU/SEI-95-TR-002). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Argimón J. (2004). "Métodos de Investigación Clínica y Epidemiológica". Elsevier España, S.A. ISBN 9788481747096.

Arisholm, E., Briand, L. C., & Foyen, A. (2004). Dynamic coupling measurement for object-oriented software. Software Engineering, IEEE Transactions on, 30(8), 491-506.

Atkinson, C., & Kuhne, T. (2003). Model-driven development: a metamodeling foundation. Software, IEEE, 20(5), 36-41.

Ball, T. (1999, January). The concept of dynamic analysis. In Software Engineering—ESEC/FSE'99 (pp. 216-234). Springer Berlin Heidelberg.

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. Software Engineering, IEEE Transactions on, 28(1), 4-17.

Barabási, A. L. (2002). Linked: How everything is connected to everything else and what it means. Plume Editors.

Baroni, A. L. (2002). Formal definition of object-oriented design metrics (Doctoral dissertation, Universidade Nova de Lisboa).

Basili, V. (1993). "The Experimental Paradigm in Software Engineering". En Experimental Software Engineering Issues: Critical Assessment and Future Directions (Ed. Rombach, H., Basili, V., Selby, R.). Lecture Notes in Computer Science, Vol. 706. ISBN 978-3-540-57092-9.

Basso, Diego M. (2014). Propuesta de métricas para proyectos de explotación de información. Tesis de Maestría en Ingeniería en Sistemas de Información. UTN. Facultad Regional Buenos Aires.

Bellini, C. G. P., Pereira, R. C. F., & Becker, J. L. (2008). Measurement in software engineering: From the roadmap to the crossroads. International Journal of Software Engineering and Knowledge Engineering, 18(01), 37-64.

Boehm, B. W. (1981). Software engineering economics (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.

Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61-72.

Booch, G., Rumbaugh, J. & Jacobson, I. (2005). The Unified Modeling Language User Guide. ISBN 9780321267979.

Burrows, R., Taïani, F., García, A., & Ferrari, F. C. (2011, June). Reasoning about faults in aspect-oriented programs: a metrics-based evaluation. In Program Comprehension (ICPC), 2011 IEEE 19th International Conference on (pp. 131-140). IEEE.

Chhabra, J. K., & Gupta, V. (2010). A survey of dynamic software metrics. Journal of computer science and technology, 25(5), 1016-1029.

Chidamber, S. R., & Kemerer, C. F. (1991). Towards a metrics suite for object oriented design (Vol. 26, No. 11, pp. 197-211). ACM.

Cho, E. S., Kim, C. J., Kim, D. D., & Rhew, S. Y. (1998, December). Static and dynamic metrics for effective object clustering. In Software Engineering Conference, 1998. Proceedings. 1998 Asia Pacific (pp. 78-85). IEEE.

Choi, K. H., & Tempero, E. (2007, January). Dynamic measurement of polymorphism. In Proceedings of the thirtieth Australasian conference on Computer science-Volume 62 (pp. 211-220). Australian Computer Society, Inc.

Churcher, N. I., Shepperd, M. J., Chidamber, S., & Kemerer, C. F. (1995). Comments on" A metrics suite for object oriented design. Software Engineering, IEEE Transactions on, 21(3), 263-265.

Cleland-Huang, J., Chang, C. K., Kim, H., & Balakrishnan, A. (2001). Requirements-based dynamic metrics in object-oriented systems. In Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on (pp. 212-219). IEEE.

Creswell, J. W. (2002). Educational research: Planning, conducting, and evaluating quantitative. Prentice Hall.

Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L., & Koschke, R. (2009). A systematic survey of program comprehension through dynamic analysis. Software Engineering, IEEE Transactions on, 35(5), 684-702.

Dorogovtsev, S. N., & Mendes, J. F. (2002). Evolution of networks. Advances in physics, 51(4), 1079-1187.

Dufour, B., Driesen, K., Hendren, L., & Verbrugge, C. (2003, October). Dynamic metrics for Java. In ACM SIGPLAN Notices (Vol. 38, No. 11, pp. 149-168). ACM.

Ejiogu, L. O. (1991). Software engineering with formal metrics. QED Information Sciences, Inc..

Erdös, P., & Rényi, A. (1959). On the central limit theorem for samples from a finite population. Publ. Math. Inst. Hungar. Acad. Sci, 4, 49-61.

Erdős, P., & Rényi, A. (1961). On the strength of connectedness of a random graph. Acta Mathematica Hungarica, 12(1), 261-267.

Erdős, P., Rényi, A., & Sós, V. T. (1966). On a problem of graph theory. Studia Sci. Math. Hungar, 1, 215-235.

Fenton, N. E., & Neil, M. (1999). Software metrics: successes, failures and new directions. Journal of Systems and Software, 47(2), 149-157.

Genero, M. (2002). Defining and validating metrics for conceptual models. Computer Science Department.

Genero, M., Piattini, M., & Calero, C. (2005). A Survey of Metrics for UML Class Diagrams. Published by ETH Zurich. Chair of Software Engineering. JOT, 2005.

Genero, M., Piattini, M., & Calero, C. (2002). Empirical validation of class diagram metrics. In Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n (pp. 195-203). IEEE.

Gilb, T. (1976). Software Metrics (Winthrop Computer Systems Series). Englewood, NJ: Winthrop.

Glass, R. L. (1994). Editor's corner A tabulation of topics where software practice leads software theory. Journal of Systems and Software, 25(3), 219-222.

Gunnalan, R., Shereshevsky, M., &Ammar, H. H. (2005). Pseudo dynamic metrics [software metrics]. In Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on (p. 117). IEEE.

Gupta, V., & Chhabra, J. K. (2011). Dynamic cohesion measures for object-oriented software. Journal of Systems Architecture, 57(4), 452-462.

Gupta, N., & Rao, P. (2001, November). Program execution based module cohesion measurement. In Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on (pp. 144-153). IEEE.

Halton, J. H. (1970). A retrospective and prospective survey of the Monte Carlo method. Siam review, 12(1), 1-63.

Halstead, M. H. (1977). Elements of software science.

Harrison, R., Counsell, S., & Nithi, R. (1997, July). An overview of object-oriented design metrics. In Software Technology and Engineering Practice, 1997. Proceedings. Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering] (pp. 230-235). IEEE.

Hassoun, Y., Johnson, R., & Counsell, S. (2004, March). A dynamic runtime coupling metric for meta-level architectures. In Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on (pp. 339-346). IEEE.

Hassoun, Y., Counsell, S., & Johnson, R. (2005, December). Dynamic coupling metric: proof of concept. In Software, IEE Proceedings- (Vol. 152, No. 6, pp. 273-279). IET.

Heitlager, I., Kuipers, T., & Visser, J. (2007, September). A practical model for measuring maintainability. In Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the (pp. 30-39). IEEE.

Henderson-Sellers, B. (1995). Object-oriented metrics: measures of complexity. Prentice-Hall, Inc.

Hitz, M., & Montazeri, B. (1996). Chidamber and Kemerer's metrics suite: a measurement theory perspective. Software Engineering, IEEE Transactions on, 22(4), 267-271.

Hossian, A. (2012). “Modelo de Proceso de Conceptualización de Requisitos” (Tesis Doctoral en Ciencias informáticas). Facultad de Informática. Universidad Nacional de La Plata.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology.

IEEE (1997). “IEEE Standard for Developing Software Life Cycle Processes. IEEE Std 1074-1997” Revision of IEEE Std 1074-1995; Replaces IEEE Std 1074.1-1995.

IEEE (1998). "IEEE Std. 1061-1998, Standard for a Software Quality Metrics Methodology, revision." Piscataway, NJ: IEEE Standards Dept., 1998.

ISO (2001). "ISO/IEC 9126-1:2001 Software engineering—Product quality—Part 1: Quality model". Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 7, Software and systems engineering.

ISO (2014). "ISO/IEC 25000:2014 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE". Revision of ISO/IEC 25000:2005. Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 7, Software and systems engineering.

Jacobson, I., Booch, G. & Rumbaugh, J (1999). The Unified Software Development Process. Addison-Wesley Object Technology.

Jiang, Y., Cuki, B., Menzies, T., & Bartlow, N. (2008, May). Comparing design and code metrics for software quality prediction. In Proceedings of the 4th international workshop on Predictor models in software engineering (pp. 11-18). ACM.

Kalos, M. H., & Whitlock, P. A. (2008). Monte Carlo methods. John Wiley & Sons.

Kaur, K., Minhas, K., Mehan, N., & Kakkar, N. (2009). Static and Dynamic Complexity Analysis of Software Metrics. World Academy of Science, Engineering and Technology, 56, 2009.

Kaner, C. & Bond W. (2004). Software engineering metrics: What do they measure and how do we know? International Software Metrics Symposium 2004. IEEE CS.

Khoshgoftaar, T. M., Munson, J. C., & Lanning, D. L. (1993, May). Dynamic system complexity. In Software Metrics Symposium, 1993. Proceedings. First International (pp. 129-140). IEEE.

Kim, H., & Boldyreff, C. (2002). Developing software metrics applicable to UML models. Proceedings of QAOOSE.

Kitchenham, B. (2010). What's up with software metrics?–A preliminary mapping study. Journal of systems and software, 83(1), 37-51.

Kleppe, A. G., Warmer, J. B., & Bast, W. (2003). MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional.

Krapivsky, P. L., Redner, S., & Leyvraz, F. (2000). Connectivity of growing random networks. Physical review letters, 85(21), 4629.

Li, W., & Henry, S. (1993, May). Maintenance metrics for the object oriented paradigm. In Software Metrics Symposium, 1993. Proceedings. First International (pp. 52-60). IEEE.

Lorenz, M., & Kidd, J. (1994). Object-oriented software metrics: a practical guide. Prentice-Hall, Inc.

Ma, Y., He, K., & Du, D. (2005). A qualitative method for measuring the structural complexity of software systems based on complex networks. In Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific (pp. 7-pp). IEEE.

Marchesi, M. (1998, March). OOA metrics for the Unified Modeling Language. In Software Maintenance and Reengineering, 1998. Proceedings of the Second Euromicro Conference on (pp. 67-73). IEEE.

Mathur, R., Keen, K. J., & Etzkorn, L. H. (2010, April). Towards an object-oriented complexity metric at the runtime boundary based on decision points in code. In Proceedings of the 48th Annual Southeast Regional Conference (p. 77). ACM.

McCabe, T. J. (1976). A complexity measure. Software Engineering, IEEE Transactions on, (4), 308-320.

Mendes, E., Mosley, N., & Counsell, S. (2005). Investigating Web size metrics for early Web cost estimation. Journal of Systems and Software, 77(2), 157-172.

Milgram, S. (1967). The small world problem. Psychology today, 2(1), 60-67.

Mitchell, A., & Power, J. F. (2004, January). An approach to quantifying the run-time behaviour of Java GUI applications. In Proceedings of the winter international symposium on Information and communication technologies (pp. 1-6). Trinity College Dublin.

Mitchell, A., & Power, J. F. (2004, June). Run-Time Cohesion Metrics: An Empirical Investigation. In Software Engineering Research and Practice (pp. 532-537).

Mitchell, A., & Power, J. F. (2005, March). Using object-level run-time metrics to study coupling between objects. In Proceedings of the 2005 ACM symposium on Applied computing (pp. 1456-1462). ACM.

Munson, J. C., & Khoshgoftaar, T. M. (1992). Measuring dynamic program complexity. Software, IEEE, 9(6), 48-55.

Myers, C. R. (2003). Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. Physical Review E, 68(4), 046116.

Oktaba, H., Garcia, F., Piattini, M., Ruiz, F., Pino, F., Alquicira, C. (2007). "Software Process Improvement: The Competisoft Project". IEEE Computer, 40(10): 21-28. ISSN 0018-9162.

OMG. (2011). 2.4. 1 superstructure specification document formal/2011-08-06. Technical report, OMG.

Ott, L., Bieman, J. M., Kang, B. K., & Mehra, B. (1995, June). Developing measures of class cohesion for object-oriented software. In Proc. Annual Oregon Workshop on Software Merics (AOWSM'95) (Vol. 11).

Page-Jones, M. (1988). The practical guide to structured systems design (Vol. 2). Englewood Cliffs, New Jersey. Yourdon Press.

Pastor-Satorras, R., & Vespignani, A. (2001). Epidemic spreading in scale-free networks. Physical review letters, 86(14), 3200.

Pressman, R. S., & Ince, D. (1992). Software engineering: a practitioner's approach (Vol. 5). New York: McGraw-Hill.

Pressman, R. S. (2005). Ingeniería del Software, 6ta edición. McGraw-Hill.

Quynh, P. T., & Thang, H. Q. (2009). Dynamic coupling metrics for service–oriented software. International Journal of Computer Science and Engineering, 3(1), 46-46.

Rational Unified Process (2001). Best practices for software development teams. A Rational Software Corporation White Paper. TP026B, Rev. 11/01.

Ravasz, E., Somera, A. L., Mongru, D. A., Oltvai, Z. N., & Barabási, A. L. (2002). Hierarchical organization of modularity in metabolic networks science, 297(5586), 1551-1555.

Roche, J. M. (1994). Software Metrics and Measurement Principles in Software Engineering Notes, ACM, vol. 19, num. 1, pp. 76 – 85.

Rodriguez, D., & Harrison, R. (2001). An Overview of Object-Oriented Design Metrics.

Rosas, L., & Riveros, H. G. (1985). Iniciación al método científico. 1ª edición.

Rothlisberger, D., Harry, M., Villazón, A., Ansaloni, D., Binder, W., Nierstrasz, O., &Moret, P. (2009, September). Augmenting static source views in IDEs with dynamic metrics. In Software Maintenance, 2009. ICSM 2009. IEEE International Conference on (pp. 253-262). IEEE.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). The unified modeling language reference manual.

Sabato, J. A., & Mackenzie, M. (1982). La producción de tecnología: autónoma o transnacional. Instituto Latinoamericano de Estudios Transnacionales.

Safari-Sharifabadi, E., & Constantinides, C. (2008). Dynamic analysis of Ada programs for comprehension and quality measurement. ACM SIGAda Ada Letters, 28(3), 15-38.

Sandhu, P. S., & Singh, G. (2008). Dynamic Metrics for Polymorphism in Object Oriented Systems. World Academy of Science, Engineering and Technology, 39.

Shannon CE, W. W. (1949). The mathematical theory of communication. Urbana.

Solé, R. V., & Valverde, S. (2004). Information theory of complex networks: On evolution and architectural constraints. In Complex networks (pp. 189-207). Springer Berlin Heidelberg.

Tahir, A., & MacDonell, S. G. (2012, September). A systematic mapping study on dynamic metrics and software quality. In Software Maintenance (ICSM), 2012 28th IEEE International Conference on (pp. 326-335). IEEE.

Torossi, G. (2015). Modelado de Objetos con UML. Cátedra de Diseño de Sistemas. Universidad Tecnológica Nacional. Facultad Regional Resistencia.

UPM, 2012. Universidad Politécnica de Madrid, Organización y Estructura de la Información. http://www.oei.eui.upm.es/Asignaturas/IS/venta_entradas/_Entradas_DRS.htm

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of ‘small-world’networks nature, 393(6684), 440-442.

Wen, L., Kirk, D., & Dromey, R. G. (2007). Software systems as complex networks. In Cognitive Informatics, 6th IEEE International Conference on (pp. 106-115). IEEE.

Wen, L., Dromey, R. G., & Kirk, D. (2009). Software Engineering and Scale-Free Networks. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 39(4), 845-854.

Yacoub, S. M., & Ammar, H. H. (2002). A methodology for architecture-level reliability risk analysis. Software Engineering, IEEE Transactions on, 28(6), 529-547.

Yacoub, S. M., Ammar, H. H., & Robinson, T. (1999). Dynamic metrics for object oriented designs. In Software Metrics Symposium, 1999. Proceedings. Sixth International (pp. 50-61). IEEE.

Zhao, M., Wohlin, C., Ohlsson, N., & Xie, M. (1998). A comparison between software design and code metrics for the prediction of software fault content. Information and Software Technology, 40(14), 801-809.

Zheng, X., Zeng, D., Li, H., & Wang, F. (2008). Analyzing open-source software systems as complex networks. Physica A: Statistical Mechanics and its Applications, 387(24), 6190-6200.