Evaluación de atributos de calidad en líneas de productos software de forma efectiva en costes

Leire Etxeberria Elorza

Mayo de 2008

Mondragon Goi Eskola Politeknikoa
Elektronika eta Informatika Saila
Mondragon Unibertsitatea

para la obtención del título de Doctor por Mondragon Unibertsitatea bajo el programa de doctorado: Nuevas Tecnologías de la Información y Comunicaciones.
Mención "doctor europeus"

dirigida por:

Dra. Goiuria Sagardui Mendieta

Zuretzat aitatxo

Resumen

La ingeniería de línea de productos software es un paradigma de desarrollo software que permite una reutilización sistemática en el desarrollo de una familia de productos, un conjunto de productos relacionados que comparten características (*features*) comunes. Una *feature* es un aspecto o característica relevante de un sistema. En la ingeniería de líneas de productos software, la validación temprana de la calidad (asegurar que los requisitos de calidad se van a lograr) cobra una importancia especial. Hay que asegurar que los activos de la línea de productos, sobre todo la arquitectura de línea de productos, sean flexibles y den soporte a todos los productos de la línea, así como a futuros productos. Sin embargo, tradicionalmente a la hora de evaluar, sólo se ha considerado la variabilidad en los requisitos funcionales y la evaluación de la variabilidad en los atributos de calidad ha recibido poca atención.

Para responder a esta problemática, se propone el método CaLiPro (Evaluación de atributos de **Ca**lidad en **Lí**neas de **Pro**ductos Software) que permite evaluar los atributos de calidad y su variabilidad en una línea de productos software de una forma efectiva en costes. Una de las bases del método es el modelado de la variabilidad en los atributos de calidad, que trata la variabilidad de los atributos de calidad como una entidad de primer grado durante todo el desarrollo y permite capturar las relaciones e interacciones existentes entre la variabilidad funcional y la variabilidad en los atributos de calidad. De este modo, es posible centrarse en los aspectos variables que afectan a la calidad y utilizar esta información para reducir el esfuerzo de evaluación y evitar tener que evaluar todos los productos uno a uno. Para ello, se proponen dos estrategias de reducción de esfuerzo de las evaluaciones: crear un modelo de evaluación genérico (con variabilidad) que permita evaluar a nivel de diseño todos los productos de la línea y seleccionar el mínimo número de productos o diseños para su evaluación de modo que se puedan extrapolar los resultados a toda la línea.

Aunque el objetivo principal del método es la evaluación de la calidad, los resultados obtenidos de la evaluación son una información muy útil durante la derivación. Por esta razón, el método también contempla una fase para facilitar una derivación de productos tendiendo en cuenta aspectos de calidad.

Abstract

Software product line engineering is a software development paradigm that allows the systematic reuse in the development of a product family, a set of related products that share common features. A feature is a relevant aspect or characteristic of a system. In software product line engineering, the quality validation (to assure that the quality requirements are met) is very important. It is necessary to ensure that the product line assets, especially the product line architecture is flexible enough to support all the products of the line and any new future product. However, traditionally, during the evaluation time only the functional requirement variability has been considered and the quality attribute variability has received little attention.

In order to overcome this limitation, the CaLiPro method to evaluate quality in software product lines has been proposed. With this method, the evaluation of the quality attributes and their variability within a software product line can be achieved in a cost-effective way. This method is based on the modeling of the quality attribute variability throughout the development phase. This modeling also helps to define the relationships and interactions between the functional and the quality attribute variability. Therefore, it is viable to predict which variable aspects affect quality in order to reduce the evaluation work and to avoid the validation of each product individually. Two strategies are proposed herein to reduce evaluation effort: 1) to create a generic evaluation model (with variability) to allow the evaluation of all the products at the design level and 2) to select the minimum number of products or designs in order to evaluate and extrapolate the results to the overall product line.

Although the main goal of this method is the evaluation, the results obtained from this evaluation can be very useful during the derivation. The method also has a phase to facilitate quality aware product derivation.

Laburpena

Software produktu lerroen ingeniaritza softwarea garatzeko paradigma bat da, eta produktu familiak sortzeko sistematikoki berrerabiltzea ahalbidetzen du. Produktu familiak elkarrekin harremana duten produktu multzoak dira, zenbait ezaugarri (*features*) batean dituztenak. Ezaugarriok sistema batean gailentzen direnak dira. Software produktu lerroen ingeniaritzan, oso garrantzitsua da kalitatea garaiz egiaztatzea: kalitate atributuak bete egingo direla ziurtatzea. Produktu lerroetako aktiboak malguak izan daitezela ziurtatu behar da, produktu lerroen egitura bereziki, eta lerroko produktu guztientzat (baita etorkizuneko produktuentzat ere) euskarri gisa balioko dutela. Alabaina, ebaluatzeko orduan irizpide funtzionalak soilik hartu ohi dira kontuan, eta kalitate atributuen aldakortasunak ez du ikertzaileen arreta bereganatzen.

Arazo horri aurre egiteko CaLiPro metodoa proposatzen dugu. Metodo horrek kalitate atributuak eta beraien aldakortasuna ebaluatzeko aukera ematen du software produktu lerroetan, modu eraginkor batean eta koste baxuarekin. Oinarrietako bat kalitate atributuen aldakortasuna modelatzea da: garapenaren une guztietan kalitatearen aldakortasuna balioestea, lehen graduko entitatetzat hartzea, alegia. Hala, aldakortasun funtzionalaren eta kalitate atributuen aldakortasunaren arteko harremanak eta interakzioak atzeman daitezke, eta posible da kalitatean eragiten duten ezaugarri aldakorrak zein diren jakitea, eta informazio hori ebaluazio ahalegina murrizteko erabiltzea, produktu denak banan-banan ebaluatu beharrik gabe. Ebaluazio ahalegina murrizteko bi estrategia proposamen dauzkagu: batetik, ebaluaketa modelo generikoa (aldakorra) sortzea; horrek produktu lerroetako produktu guztiak ebaluatzea ahalbidetuko du. Eta bestetik, ahalik eta produktu edo diseinu kopuru txikiena aukeratzea ebaluaziorako, eta gero emaitzak lerro osora estrapolatzea.

Metodoaren helburu nagusia kalitatea ebaluatzea bada ere, lortutako emaitzak oso baliagarriak dira deribazio aldian ere. Horregatik, kalitate irizpideak aintzat hartuta produktuen deribazioa errazteko fase bat ere aurreikusten da.

Agradecimientos

Hitz batzuk eskaini nahiko nizkieke lan hau egiten ari nintzen bitartean lagundu didaten pertsona guztiei. Lehendabizi Goiuriari, lanean aritu garen urte hauetan izan duen jarrera eskertzeko: bere ideiak, pazientzia eta laguntza.

Nire familiari, bereziki amari, Idoiari, Nereri eta Amaiari beraien laguntza eta sosteguarengatik. Alde batetik hain garrantsitzua den babes emozionala emateagatik, eta baita ere, gaia ulertu ez arren, irakurri eta laguntzen saiatu direlako. Eta lagunei ere bai, erakutsi duten interesarengatik.

Nire lankideak ere ez ditut ahazten, oraingoak eta lehengoak: Lorea, Gentzane, Xabi eta "Hazi" taldeko kide guztiak. Eta, noski, informatika departamentuko eta unibertsitateko lankideak. Seguruenez norbait ahaztuko dut. Horrenbestez, aldez edo moldez lagundu didaten guztiei eskerrik beroenak.

También agradezco la ayuda de las personas que han revisado esta tesis y me han ayudado a mejorarla: Salvador Trujillo, Roberto López-Herrejón y Pascal Poizat. Finalmente, esta tesis ha sido posible gracias a la financiación obtenida a través de la beca BFI04.136 del Programa de Formación de Investigadores del Departamento de Educación, Universidades e Investigación del Gobierno Vasco.

Tabla de Contenidos

1. Introducción **1**

1.1. Introducción 1

1.2. Objetivo e hipótesis 5

1.3. Contribuciones 6

1.4. Método de investigación 8

1.5. Organización del documento 9

2. Estado del arte **13**

2.1. Introducción 13

2.2. Línea de productos software 14

2.3. Atributos de calidad 19

2.4. Aseguramiento de calidad en líneas de productos software 21

2.5. Conclusiones 22

3. Modelado y evaluación de la calidad en las líneas de productos **23**

3.1. Introducción 23

3.2. Atributos de calidad en las líneas de productos 23

3.3. Modelado de la variabilidad en atributos de calidad 26

3.4. Evaluación de atributos de calidad en líneas de productos 30

3.5. Conclusiones 38

4. Visión general del método CaLiPro (Evaluación de atributos de Calidad en Líneas de Productos) **41**

4.1. Introducción 41

4.2. Evaluación de la calidad 44

4.3. Modelado de características extendido con atributos de calidad . 47
4.4. Herramientas desarrolladas 53
4.5. Encaje con metodologías existentes 56
4.6. Conclusiones 56

5. Modelo CaFM (Modelo de características extendido con atributos de Calidad) 59

5.1. Introducción 59
5.2. Modelo CaFM 60
5.3. Características de calidad 61
5.4. Impactos 64
5.5. Conclusiones 66

6. Proceso CaLiPro 69

6.1. Introducción 69
6.2. Fase Especificación de la Variabilidad 71
6.3. Fase Validación de calidad 72
6.4. Fase Derivación 88
6.5. Conclusiones 93

7. Casos de estudio 95

7.1. Introducción 95
7.2. Descripción de los casos de estudio 98
7.3. Conclusiones globales de los casos de estudio 103

8. Conclusiones y líneas futuras 105

8.1. Conclusiones 105
8.2. Resultados 110
8.3. Limitaciones 112
8.4. Líneas futuras 113

Referencias bibliográficas 115

Glosario 137

Anexos **139**

A. Conclusions and future work **139**

A.1. Conclusions 139

A.2. Results 144

A.3. Limitations 146

A.4. Future work 147

B. Comparativa de enfoques de modelado de características **149**

B.1. FODA 149

B.2. FORM 150

B.3. FeatuRSEB 151

B.4. *Generative Programming* 152

B.5. Enfoque de Van Gurp 152

B.6. Enfoque de Riebisch 153

B.7. CBFM 154

B.8. PLUSS 155

B.9. Los atributos de calidad en los enfoques 155

B.10. Resumen 155

C. Descripción SPEM del proceso CaLiPro **159**

C.1. Introducción 159

C.2. El proceso CaLiPro 161

C.3. El modelo CaFM 163

C.4. Activos de la línea de productos 163

C.5. Fases del proceso CaLiPro 164

D. Extensión del árbol de utilidad **199**

D.1. Introducción 199

D.2. *Quality feature tree*: Extensión del árbol de utilidad 200

E. Comparativa de métodos de evaluación de arquitecturas 203

E.1. Métodos generales 204

E.2. Relacionados con la evolución: modificabilidad, mantenibilidad, flexibilidad, etc. 205

E.3. Rendimiento (*Performance*) 208

E.4. Fiabilidad (*Reliability*) 210

E.5. Usabilidad 213

F. Selección de productos y detección de interacciones 215

F.1. Introducción 215

F.2. Interacciones o impactos 216

F.3. Algoritmo de selección de productos 218

F.4. Detectar interacciones 220

G. Caso Línea de Productos Juegos Arcade 225

G.1. Introducción 225

G.2. Especificación de la Variabilidad 227

G.3. Validación de la calidad 228

G.4. Derivación 232

G.5. Conclusiones 236

H. Caso Línea de Productos Calculadora Software 237

H.1. Introducción 237

H.2. Especificación de la Variabilidad 238

H.3. Validación de la calidad 240

H.4. Derivación 248

H.5. Enfoque *bottom-up* 251

H.6. Conclusiones 253

I. Caso GPL (Graph Product line) 255

I.1. Introducción 255

I.2. Especificación de la variabilidad 257

I.3. Validación de calidad 259

I.4. Derivación 263
I.5. Enfoque *bottom-up* 264
I.6. Conclusiones 265

Índice de figuras

1.1. Estructura de los capítulos 10

3.1. Clasificación de los requisitos de línea de productos 24
3.2. Momentos de evaluación en el contexto de una línea de productos 32

4.1. Fases del proceso CaLiPro 42
4.2. Ejemplo del enfoque FODA 49
4.3. Imagen de pantalla de la herramienta fmp-CaLiPro 54
4.4. Imagen de pantalla de herramienta guiCa 55

5.1. Metamodelo del modelo CaFM 60
5.2. Categorías y subcategorías del modelo CaFM 62
5.3. Ejemplo de un *quality feature tree* 64

6.1. Proceso completo 70
6.2. Modelo CaFM de la línea de productos de coches 72
6.3. Modelo genérico: El gráfico de ejecución para el escenario de refresco 76
6.4. Interacciones detectadas en el ejemplo del coche 87

7.1. Modelo CaFM de la LP Arcade 99
7.2. Modelo CaFM de la LP calculadora 100
7.3. Modelo CaFM del GPL 102

B.1. Ejemplo del enfoque FODA 150

C.1. Fases del proceso CaLiPro 160
C.2. Proceso CaLiPro 161

C.3. Diagrama de actividad que describe las fases y productos de trabajo del proceso CaLiPro 162
C.4. Diagrama de clases del modelo CaFM 164
C.5. Diagrama de clases de los activos de la línea de productos 165
C.6. Diagrama de casos de uso con las definiciones de trabajo que incluye la fase de especificación de la variabilidad 166
C.7. Diagrama de actividad que describe la fase de especificación de la variabilidad en términos de definiciones de trabajo y productos de trabajo 166
C.8. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Análisis de la variabilidad en los requisitos" . 167
C.9. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Análisis de la variabilidad en el diseño" 167
C.10. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Análisis de la variabilidad en la implementación" 168
C.11. Diagrama de actividad de la definición de trabajo "Análisis de la variabilidad en los requisitos" 171
C.12. Diagrama de actividad de la definición de trabajo "Análisis de la variabilidad en el diseño" 172
C.13. Diagrama de actividad de la definición de trabajo "Análisis de la variabilidad en la implementación" 174
C.14. Diagrama de casos de uso con las definiciones de trabajo que incluye la fase de validación 176
C.15. Diagrama de actividad que describe la fase de validación en términos de definiciones de trabajo y productos de trabajo . . . 177
C.16. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Seleccionar atributos de calidad y forma de evaluar" 177
C.17. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Crear modelo de evaluación genérico" 178
C.18. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Seleccionar productos a evaluar" 178
C.19. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Evaluación de los productos" 179
C.20. Diagrama de actividad de la definición de trabajo "Seleccionar atributos de calidad y forma de evaluar" 182

C.21. Diagrama de actividad de la definición de trabajo "Crear modelo de evaluación genérico" 185
C.22. Diagrama de actividad de la definición de trabajo "Seleccionar productos a evaluar" 187
C.23. Diagrama de actividad de la definición de trabajo Evaluación de los productos 190
C.24. Diagrama de casos de uso con las definiciones de trabajo que incluye la fase de derivación 192
C.25. Diagrama de actividad que describe la fase de derivación en términos de definiciones de trabajo y productos de trabajo . . . 193
C.26. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Preparar para la derivación" 193
C.27. Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Uso del CaFM durante la derivación" 194
C.28. Diagrama de actividad de la definición de trabajo "Preparar para la derivación" 195
C.29. Diagrama de actividad de la definición de trabajo "Uso del CaFM durante la derivación" 197

D.1. Árbol de utilidad original versus QFT (*Quality Feature Tree*) . . 201

G.1. Ejemplo de una pantalla del juego *SpaceWar* 226
G.2. Modelo CaFM de la LP Arcade 227
G.3. El gráfico de ejecución para el escenario de refresco 229
G.4. Modelo CaFM de la LP Arcade con pesos y niveles 234

H.1. Ejemplo de calculadora en el emulador de Sony Ericsson 990 . . 238
H.2. Modelo CaFM de la LP calculadora 239
H.3. Características de calidad del modelo CaFM de la calculadora para derivación 252

I.1. Modelo de CaFM de GPL en diseño e implementación 258

Lista de Tablas

3.1. Enfoques para representar variabilidad en los atributos de calidad 29

6.1. Sobrecarga de procesamiento 77
6.2. Matriz de trazabilidad de la variabilidad 77
6.4. Productos seleccionados y consumo de dichos productos 86
6.5. Diferentes escenarios durante la derivación 91
6.6. Diferentes situaciones durante la derivación 92

7.1. Resumen de los casos de estudio 104

B.1. Enfoques basados en modelos de características 156

C.1. Resumen de las actividades de la fase de Especificación de la variabilidad 168
C.2. Resumen de las actividades de la fase de Validación 180
C.3. Resumen de las actividades de la fase de Derivación 192

D.1. Diferencias entre el árbol de utilidad original y la extensión propuesta 202

E.1. *Framework* de clasificación de métodos de evaluación 204
E.2. Métodos de evaluación de arquitecturas software genéricos . . . 205
E.6. Método de evaluación para usabilidad 213

G.1. Impactos cualitativos 228
G.2. Sobrecarga de procesamiento 231
G.3. Recursos de ordenador totales por paso de procesamiento 231
G.4. Recursos de ordenador totales teniendo en cuenta las repeticiones 231

G.5. Matriz de trazabilidad de la variabilidad 232
G.6. Impactos cuantitativos sobre tiempo de refresco en milisegundos 233
G.7. Impactos cuantitativos sobre usabilidad 235

H.3. Rangos para 3 grupos o niveles en usabilidad 250
H.4. Rangos para 3 grupos o niveles en *accuracy* 251

I.1. Productos y resultados de la medición de los productos en milisegundos 262

Capítulo 1

Introducción

1.1. Introducción

En todos los productos, sean desarrollados de forma individual o mediante paradigmas de desarrollo como las líneas de producto software, la calidad del software es muy importante ya que de ello depende la satisfacción de los *stakeholders* del producto: cliente, usuario final, encargado de mantenimiento, etc. La calidad software es el grado en el que el software posee una combinación de atributos deseada [IEEE 93]: Rendimiento, seguridad, disponibilidad, usabilidad, funcionalidad, modificabilidad, mantenibilidad, etc. La calidad software suele ser transparente cuando está presente, pero resulta fácilmente reconocible cuando está ausente. Muchas veces existen requisitos de calidad implícitos (usabilidad, mantenibilidad, etc.) que si no se alcanzan ponen en entredicho la calidad del software. En otras ocasiones suelen ser explícitos, como ciertos atributos que cobran especial importancia en algunos dominios de aplicación del software. Por ejemplo, los aspectos de tiempo y rendimiento en los sistemas empotrados de tiempo real o los aspectos de fiabilidad o *safety* en los sistemas empotrados críticos para la seguridad (*safety-critical*).

Una línea de productos software permite una reutilización sistemática en los casos en los que se tiene una familia de productos, es decir, productos similares, diferenciados por algunas características. Ciertamente, ésta es una situación muy común, por lo que la ingeniería de líneas de productos es aplicable en un gran porcentaje de desarrollos software obteniéndose varias ventajas [Pohl 05] como reducir los costes de desarrollo, aumentar la calidad, reducir el tiempo de entrega de los productos (*Time to Market*), reducir el esfuerzo de mantenimiento, lidiar con la evolución, lidiar con la complejidad o mejorar la

estimación de costes. Además, los clientes también se benefician con productos adaptados a sus necesidades.

En el caso de las líneas de productos software, validar y asegurar la calidad software durante el desarrollo es especialmente importante debido al alcance de una línea de productos que puede abarcar muchos productos. Por lo tanto, un error o una decisión de diseño inadecuada pueden propagarse a muchos productos. Sin embargo, también es más complicado que en sistemas individuales debido a la variabilidad presente en la línea.

En una línea de productos, existen dos tipos de atributos de calidad a considerar: los atributos de calidad específicos de línea de productos y los atributos de calidad relevantes del dominio. Los atributos específicos de línea de productos son aquellos relacionados con la variabilidad o flexibilidad. La variabilidad [Thiel 02] entendida como modificabilidad (para permitir la variación o evolución en el tiempo) y configurabilidad (variabilidad en el espacio) para obtener un conjunto de productos relacionados. Por ello, evaluando la variabilidad, se asegura que utilizando los activos de la línea es posible lograr toda la funcionalidad de todos los productos dentro del alcance definido para la línea. Los atributos de calidad relevantes de dominio están relacionados con el dominio específico de la línea de productos (tales como la seguridad en dominios donde la seguridad es crítica, rendimiento en dominios de tiempo real, fiabilidad en sistemas empotrados, etc.). Puesto que los diferentes productos de un dominio pueden requerir diferentes valores en los atributos (por ejemplo, no todos los productos requieren el mismo nivel de seguridad), la variabilidad en la manera en que el atributo se traslada al producto es relevante durante la evaluación; para asegurar que se pueden obtener todos los atributos de calidad para todos los productos definidos en el alcance de la línea. La evaluación de la calidad de la línea consiste en validar el abanico de posibles niveles de calidad de los productos de la línea.

La evaluación de un sistema para comprobar si cumple los atributos de calidad requeridos puede realizarse a diferentes niveles de abstracción: diseño (evaluación de la arquitectura software), implementación (testeos), etc. La arquitectura software de un sistema tiene una gran influencia en la calidad final de los sistemas ya que puede inhibir o permitir los atributos de calidad de los productos. La evaluación de la arquitectura software se define como "el examen sistemático de la extensión con la que una arquitectura software cumple los requisitos" [Dolan 01]. Estos requisitos pueden ser tanto funcionales como atributos de calidad. La calidad software debe ser inherente desde el principio e incorporado mediante el diseño, ya que no se puede añadir posteriormente [Bass 98]. De ahí la importancia de evaluar los atributos de calidad en la arquitectura software. Además, analizar el potencial de una arquitectura para

obtener los niveles de calidad requeridos ayuda a encontrar errores de una forma temprana en el ciclo de vida, cuando son más sencillos y baratos de corregir que en etapas posteriores como la implementación, el testeo o el despliegue. La calidad debe ser considerada en todas las fases de diseño, implementación y despliegue. Por lo tanto, la evaluación de la calidad no sólo se puede centrar en el diseño, también es necesario evaluar el código.

Uno de los desafíos a afrontar en la ingeniería de línea de productos software es la falta de técnicas y métodos para asegurar la calidad software. La mayoría de los métodos existentes para asegurar la calidad en sistemas únicos no soportan la variabilidad, mientras que la mayoría de métodos creados específicamente para las líneas de productos se centran en los atributos relacionados con la flexibilidad; pero no dan soporte adecuado para evaluar los atributos de dominio. En una línea de productos debido a la variabilidad en estos atributos de calidad, para poder evaluar que los atributos se cumplen, hay que asegurarse de que se cumplan los valores requeridos para todos los productos. Esto dificulta la aplicación de métodos de aseguramiento de calidad (originalmente pensados para un solo sistema) y en algunos casos conlleva evaluar todos los productos de la línea con el coste que esto supone.

Sin embargo, se puede seguir la siguiente premisa: "La evaluación de la arquitectura de producto es una variación de la evaluación de la arquitectura de línea de productos como la arquitectura de productos es una variación de la arquitectura de línea de productos y la extensión en la que la evaluación de producto es una evaluación separada y dedicada depende de la extensión en la que la arquitectura de producto se diferencia en formas que afectan a los atributos de calidad de la arquitectura de línea de productos" [Clements 01]. La presente tesis propone el método CaLiPro (Evaluación de **Ca**lidad en **Lí**neas de **Pro**ductos Software) que se basa en esta premisa para facilitar la evaluación de la calidad en una línea de productos de una forma efectiva en coste. El método es aplicable tanto a nivel de arquitectura software como a nivel de implementación.

El método propuesto se basa en modelar la variabilidad en los atributos de calidad (variabilidad en los atributos de calidad de dominio). Esta variabilidad se consigue a través de la variabilidad en el diseño y la implementación de la línea. Sin embargo, por el hecho de ser consecuencia u obtenerse gracias a la variabilidad en el diseño o la implementación, la variabilidad en los atributos de calidad no ha sido tratada como una entidad de primer orden y se ha mantenido en segundo plano durante mucho tiempo. No obstante, en los últimos años los investigadores dedican a este problema cada vez más tiempo [Halmans 03] [Niemelä 07] y han observado que la variabilidad en los atributos de calidad, debido a su naturaleza, no se puede modelar utilizando los méto-

dos y modelos existentes hasta ahora, ya que estos enfoques se basan en la variabilidad funcional [Myllärniemi 06].

Para el modelado, se utiliza un enfoque de modelado de características extendido (llamado CaFM: **Ca**LiPro **F**eature **M**odel). El modelo CaFM se utiliza en la ingeniería de dominio: durante el análisis, diseño e implementación del dominio para capturar la variabilidad y posteriormente durante la evaluación. El modelado de características es un enfoque muy utilizado para representar la variabilidad en las líneas de productos software y ha sido extendido para representar la variabilidad en los atributos de calidad. El modelo permite especificar cómo impacta la variabilidad funcional, arquitectural y de implementación en la variabilidad en los atributos de calidad. De este modo, el modelo de la variabilidad se puede utilizar como fuente de información para centrarse en los aspectos que afectan a la calidad y de este modo reducir el esfuerzo de evaluación, y evitar tener que evaluar todos los productos uno a uno.

El modelado de la variabilidad tanto funcional como de atributos de calidad es el punto de partida de las dos estrategias de reducción de esfuerzo de las evaluaciones que se proponen. La primera de ellas consiste en crear un modelo de evaluación genérico (con variabilidad) que permita evaluar a nivel de diseño todos los productos de la línea. La segunda estrategia consiste en seleccionar el mínimo número de productos o diseños para su evaluación, de modo que se puedan extrapolar los resultados a toda la línea. Estas dos estrategias también pueden ser complementarias: primeramente se genera un modelo genérico y sólo se instancia para los productos seleccionados.

El método está pensado para evaluaciones tanto a nivel de diseño (evaluación de la arquitectura) como a nivel de implementación (ejecución y medición de aspectos de calidad). Además, permite reducir el esfuerzo de evaluación y, por lo tanto, el coste en hasta un 98 % respecto a evaluar los productos uno a uno.

Aunque el objetivo principal del método es la evaluación de la calidad, los resultados obtenidos de la evaluación son una información muy útil durante la derivación de productos: "El proceso completo de construir un producto a partir de los activos software de la línea de productos [Deelstra 05]". Por esta razón, el método también contempla añadir al modelo CaFM información relevante para la derivación obtenida a partir de los resultados de la evaluación. De este modo, se puede realizar una derivación de productos tendiendo en cuenta aspectos de calidad.

Debido a la falta de técnicas de modelado de la variabilidad que consideren los atributos de calidad, a menudo se realiza la derivación de los productos sin tener información de cómo las diferentes opciones que se pueden seleccionar

afectan a la calidad y sin poder seleccionar los niveles de calidad deseados cuando éstos son variables. Con el uso del modelo CaFM aumentado con los resultados de la evaluación, es posible realizar una evaluación teniendo en cuenta la calidad; ya que será posible saber cuánto aumenta el nivel de un atributo de calidad al seleccionar una opción.

Por lo tanto, el modelo CaFM se utiliza no sólo durante la ingeniería de dominio sino también durante la ingeniería de aplicación para facilitar una derivación guiada por atributos calidad.

1.2. Objetivo e hipótesis

El objetivo principal ha consistido en desarrollar un método para evaluar los atributos de calidad de una línea de productos software de una forma efectiva en costes. El método sirve para evaluar atributos de calidad relevantes del dominio tanto a nivel de diseño como de implementación. El método permite reducir el esfuerzo de evaluación mediante estrategias que utilizan la información de la variabilidad en los atributos de calidad. Para ello es necesario tratar la variabilidad de los atributos de calidad como una entidad de primer grado durante el diseño.

Los objetivos concretos que se persiguen para la consecución del objetivo general son:

1. Definición de una técnica para modelar la variabilidad que incluya la variabilidad en los atributos de calidad, en especial las relaciones de influencia entre la variabilidad funcional y la variabilidad en los atributos de calidad.
2. Desarrollo de técnicas y estrategias que permitan hacer uso del modelado de variabilidad para reducir el esfuerzo de evaluación.
3. Desarrollo de técnicas para facilitar la derivación de productos teniendo en cuenta los atributos de calidad y haciendo uso de los resultados de la evaluación.

Las hipótesis de la investigación han sido las siguientes:

- A partir de la premisa: "La evaluación de la arquitectura de producto es una variación de la evaluación de la arquitectura de línea de productos como la arquitectura de productos es una variación de la arquitectura de

línea de productos y la extensión en la que la evaluación de producto es una evaluación separada y dedicada depende de la extensión en la que la arquitectura de producto se diferencia en formas que afectan a los atributos de calidad de la arquitectura de línea de productos" [Clements 01], la hipótesis es la siguiente: modelando la variabilidad en los atributos de calidad y su relación con la variabilidad funcional, arquitectural y de implementación, y utilizando esta información es posible reducir el esfuerzo de la evaluación de la calidad en una línea de productos.

- Los resultados de la evaluación de la calidad y la variabilidad modelada proporcionan información que permite realizar una derivación teniendo en cuenta los atributos de calidad.

Dentro de estas dos hipótesis principales han surgido nuevas hipótesis que también han sido validadas:

- Los atributos de calidad y su variabilidad pueden representarse utilizando modelos de características extendidos.
- Existen interacciones entre las características cuando impactan en los atributos de calidad. El grado de interacción de las características influye en el esfuerzo a realizar para evaluar la calidad. Cuanto mayor es el grado de interacción, mayor será el esfuerzo para evaluar ese atributo de calidad.

1.3. Contribuciones

Dentro del contexto explicado en la sección anterior, esta investigación responde principalmente al siguiente problema:

- Evaluación de la calidad en líneas de producto software.
  - Problema: La evaluación de la calidad en las líneas de productos software es costosa debido a la variabilidad y existen muy pocos métodos para evaluar atributos de calidad relevantes del dominio en las líneas de productos, por lo que en el caso de algunos atributos suele ser necesario evaluar todos los productos.
  - Contribución: Método de evaluación de atributos de calidad que permite reducir el esfuerzo de evaluación en hasta un 98 % respecto a evaluar los productos uno a uno.

Por lo tanto, la principal contribución de la investigación ha consistido en desarrollar un método que facilita la evaluación de la calidad reduciendo el esfuerzo a realizar y que es aplicable tanto a nivel de diseño como a nivel de implementación. Para lograr esta contribución o complementarla, también se han trabajado los siguientes aspectos:

- Estudio de la evaluación de los atributos de calidad en líneas de productos software.
  - Problema: La evaluación de atributos de calidad para sistemas únicos es una área madura. Sin embargo, la evaluación de la calidad en las líneas de productos implica nuevos retos y aspectos a considerar que no han sido analizados en profundidad.
  - Contribución: Se ha realizado un análisis de la evaluación de atributos de calidad en líneas de productos: se ha realizado una clasificación de atributos de calidad específica para las líneas de productos software, se han detectado nuevos aspectos a considerar en la evaluación, así como nuevos momentos en los que la evaluación puede realizarse. También se ha realizado un estudio y comparativa de métodos y métricas de evaluación de sistemas únicos que se pueden aplicar en las líneas de productos software, así como de los métodos específicos para las líneas de productos.
- Modelado y gestión de la variabilidad en atributos de calidad en las líneas de productos software.
  - Problema: La variabilidad en los atributos de calidad, por el hecho de ser a menudo consecuencia de la variabilidad en la implementación, no ha sido tratado como una entidad de primer orden y no existen modelos y métodos que cubran todas las necesidades de este tipo de variabilidad.
  - Contribución: Estudio de la variabilidad en los atributos de calidad y de las relaciones e impactos entre la variabilidad funcional y la variabilidad en los atributos de calidad, estudio y comparativa de técnicas de modelado de variabilidad y desarrollo de un modelo de variabilidad basado en el modelo de características que permite gestionar la variabilidad de los atributos de calidad como una entidad de primer grado durante el diseño, evaluación y derivación en las líneas de productos software.

- Interacción de las características y su relación con la calidad.
  - Problema: Las características funcionales de una línea de productos pueden afectar a los atributos de calidad. Además las características pueden interaccionar entre ellos causando impactos diferentes en la

calidad. Se ha comenzado a investigar en las interacciones de características pero no existe un estudio del efecto de estas características en la calidad.

- Contribución: Se ha realizado un estudio de las interacciones entre características y su efecto en la calidad en una línea de productos y se ha definido un enfoque que ayuda a detectar estas interacciones.

- Derivación de productos en las líneas de productos.
  - Problema: Actualmente, las técnicas de modelado de la variabilidad no consideran explícitamente los atributos de calidad por lo que la derivación de productos se realiza sin una información completa sobre cómo la selección de características funcionales afecta a los niveles de calidad.
  - Contribución: Método que facilita una derivación guiada por atributos de calidad o con información de los niveles de atributos de calidad que se van obteniendo durante la derivación. De este modo es posible seleccionar el nivel de cierto atributo de calidad para un producto y no sólo las funcionalidades que va a tener el producto.

1.4. Método de investigación

Existen muchas clasificaciones de metodologías de investigación [Barchini 05] [Myers 97]. Una de las clasificaciones más comunes es la que divide los métodos en lógicos, hipotético-deductivos y empíricos. Los métodos lógicos hacen referencia a las metodologías que utilizan el pensamiento o deducciones para llegar al conocimiento. El método hipotético-deductivo se basa en la propuesta de una hipótesis como consecuencia de procedimientos inductivos. Se llega a conclusiones particulares a partir de las hipótesis que luego se comprueban experimentalmente. Por último, los métodos empíricos hacen referencia a un conjunto de metodologías donde se aproxima al conocimiento del objeto y directamente se hace uso de la experiencia.

Dentro de los métodos empíricos, los métodos se pueden clasificar en cuantitativos y cualitativos. Los métodos cuantitativos son los que se desarrollan en las ciencias naturales o en las ciencias del estudio de fenómenos naturales. Algunos ejemplos de métodos cuantitativos incluyen los métodos encuesta, experimentos de laboratorio, métodos formales (p.e.: econométricas) y métodos numéricos tales como el modelado matemático. Por el contrario, los métodos

cualitativos se desarrollaron en las ciencias sociales y son investigaciones del estudio social y de los fenómenos culturales.

En nuestro caso, el método de investigación que se ha seguido durante la realización de la tesis ha sido el "caso de estudio", que se puede clasificar como un método empírico y cualitativo. El método "caso de estudio" está particularmente bien adaptado a la investigación en la Ingeniería del Software y los Sistemas de Información [Myers 97].

"Un caso de estudio es una indagación empírica que investiga un fenómeno contemporáneo en el contexto de su vida real, especialmente cuando las fronteras entre el fenómeno y el contexto no son evidentes" [Yin 02]

Investigación de "caso de estudio", son aquellos casos de estudios únicos o múltiples y pueden incluir evidencias cuantitativas, confiar en múltiples fuentes de evidencia y se benefician de desarrollos de proposiciones teóricas anteriores. En vez de utilizar grandes muestras y seguir un protocolo rígido para examinar un número de variables limitados, los métodos "caso de estudio" realizan un examen longitudinal y en profundidad de una sola instancia o evento: un caso. Proporcionan una forma sistemática de mirar eventos, recoger datos, analizar información y realizar informes de resultados. Como resultado, el investigador puede lograr un conocimiento más profundo de por qué la instancia se comportó así, y qué puede ser importante para mirar más extensamente en investigaciones futuras. Los casos de estudio permiten ambas cosas: generar y testear hipótesis. El método de investigación "caso de estudio" puede basarse en una mezcla de evidencias cuantitativas y cualitativas.

Este tipo de método ha funcionado bien con la investigación realizada, ya que ha permitido probar las hipótesis y la validez del método desarrollado. En nuestro caso, se han utilizado tres casos de estudio que han permitido validar las hipótesis y también generar nuevas hipótesis que han sido posteriormente validadas. Dos de los casos de estudio se basan en casos de líneas de productos software públicos que se han definido para el aprendizaje y la experimentación.

1.5. Organización del documento

Este documento está dividido en siete capítulos principales, además de esta introducción (ver figura 1.1). A partir de la problemática que se detalla en este capítulo, en el capítulo 2 se presenta el estado del arte de las áreas de

la tesis, en el capítulo 3 se realiza un estudio de las técnicas y métodos de modelado y evaluación de los aspectos de calidad en las líneas de productos, en los capítulos 4, 5 y 6 se presenta el método que se propone para responder a la problemática, en el capítulo 7 se presentan los casos de estudio llevados a cabo para aplicar el método propuesto y en el capítulo 8 se presentan las conclusiones y las líneas futuras.

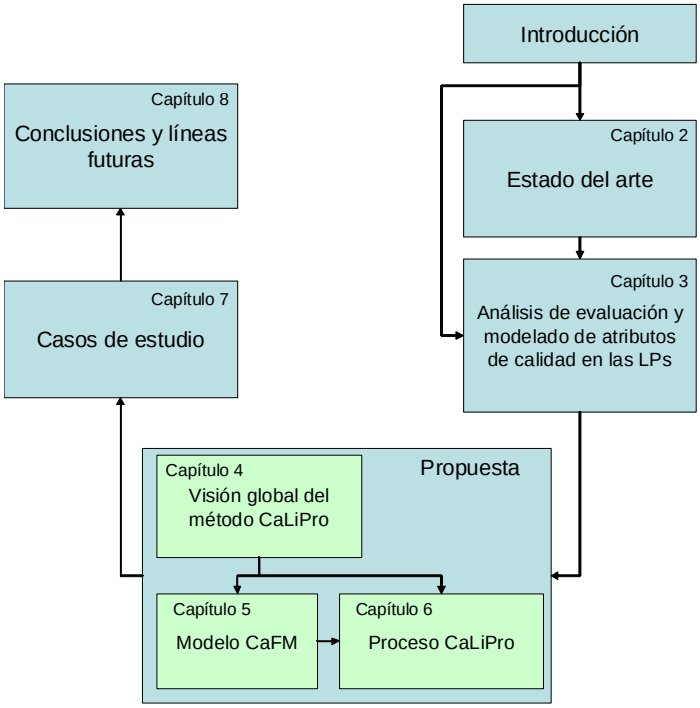


Figura 1.1: Estructura de los capítulos

Capítulo 1: Introducción. Este capítulo introduce brevemente los temas principales de la tesis, la problemática, las contribuciones realizadas y el enfoque de la investigación.

Capítulo 2: Estado del arte. Este capítulo presenta una visión global de las áreas principales de la tesis: líneas de productos software, variabilidad, arquitecturas software, atributos de calidad, aseguramiento de la calidad en contextos de reutilización, etc. El lector puede saltarse este capítulo si está familiarizado con estos conceptos.

Capítulo 3: Modelado y evaluación de la calidad en líneas de productos software. Análisis de los atributos de calidad y su variabilidad en las líneas de productos software. Así como de los métodos y técnicas existentes para la eva-

luación de líneas de productos y modelado de la variabilidad de los atributos de calidad.

Capítulo 4: Visión general del método CaLiPro de evaluación de atributos de calidad en líneas de productos software. Capítulo que introduce el método propuesto para facilitar la evaluación de líneas de productos y la derivación de productos así como las herramientas desarrolladas para dar soporte al mismo.

Capítulo 5: Modelo CaFM. Capítulo que describe el modelo que se propone para especificar la variabilidad en los atributos de calidad y los impactos entre la variabilidad funcional y la variabilidad en la calidad.

Capítulo 6: Proceso CaLiPro. Capítulo que describe el proceso de aplicación del método CaLiPro para facilitar la evaluación de líneas de productos y la derivación de productos.

Capítulo 7: Casos de estudio. Este capítulo describe los resultados de tres casos de estudio llevados a cabo para validar la reducción de esfuerzo que supone la aplicación del método. En concreto los tres casos de estudio son el *Arcade Game Maker (AGM) product line*, la línea de productos Calculadora Software y el *Graph Product line*.

Capítulo 8: Conclusiones y líneas futuras. Este capítulo resume las conclusiones recopiladas, así como líneas futuras de la tesis.

Además de los capítulos mencionados, el documento se compone de un apartado con el glosario que define los términos utilizados en la tesis y otro apartado con las referencias. Así como varios anexos con información más completa de algunos de los aspectos a los que se hace referencia desde los capítulos.

Capítulo 2

Estado del arte

2.1. Introducción

El presente capítulo describe el estado del arte de las áreas en las que se enclava la tesis: las líneas de productos software, los atributos de calidad y el aseguramiento de la calidad.

El paradigma de la ingeniería de línea de productos es introducido proporcionando información de su definición, beneficios y desventajas, activos núcleo, las fases dentro de este paradigma, la variabilidad implícita de la misma y las técnicas existentes para modelarla, así como enfoques y herramientas de ingeniería de línea de productos.

Respecto a los atributos de calidad, se definen los conceptos de calidad software y atributos de calidad y se detallan las clasificaciones de atributos de calidad y la naturaleza de los atributos de calidad.

También se describe el aseguramiento de la calidad en contextos de reutilización.

2.2. Línea de productos software

"Una **línea de productos software** es un conjunto de sistemas software intensivos que comparten un conjunto de características que satisfacen las necesidades específicas de un segmento de mercado particular y que son desarrollados a partir de un conjunto de activos núcleo en un modo preestablecido" [Clements 01]. Las líneas de productos permiten obtener beneficios de las partes comunes de los distintos productos que desarrolla una organización a la vez que limita la variabilidad entre los mismos. **Las líneas de productos permiten una reutilización sistemática en los casos en los que se tienen familias de productos,** es decir productos similares, diferenciados por algunas características.

Las organizaciones que han aplicado la ingeniería de línea de productos han logrado beneficios remarcables que están alineados con los objetivos de negocio [Clements 01]: ganancias en productividad, disminución de tiempo de entrega de los productos (*Time to Market*), aumento de la calidad de los productos, disminución de los riesgos, incremento de la agilidad en el mercado, incremento de la satisfacción del cliente, uso más eficiente de los recursos humanos, habilidad para efectuar personalizaciones en masa, habilidad para mantener la presencia en el mercado, habilidad para sostener un crecimiento sin precedentes, etc.

Estos beneficios dan a las organizaciones una ventaja competitiva y se derivan de la reutilización de los activos núcleos de forma estratégica y preestablecida. Una vez que la base de los activos núcleos para la línea de productos se ha establecido, hay un ahorro directo cada vez que se construye un producto.

El enfoque de línea de productos software también proporciona opciones frente a oportunidades de mercado futuras. Aunque las oportunidades exactas y su probabilidad no se pueden predecir, haciendo uso de los puntos de variación de los activos núcleos, las líneas de productos permiten realizar experimentos con coste y riesgo bajo para explorar oportunidades. Y otro de los aspectos importantes es que las líneas de productos aumentan la calidad. Cada nuevo sistema se beneficia de la eliminación de los defectos de los productos anteriores; por lo que la confianza del desarrollador y del cliente crece en cada nueva instalación.

El enfoque de línea de productos también tiene sus riesgos e inconvenientes. Una de las desventajas principales es que requiere grandes inversiones, supone un riesgo y tiene un impacto importante tanto a nivel de organización como de gestión en términos de cambio de mentalidad y costes de tiempo y dinero. Es por ello que debe ser una estrategia a medio-largo plazo y no esperar resultados

inmediatos. Las líneas de productos no aparecen accidentalmente, si no que requieren de un esfuerzo explícito y consciente para poder iniciar una línea de productos.

2.2.1. Activos núcleo

Los activos núcleo son los principales elementos de una línea de productos ya que recopilan toda la experiencia y el conocimiento de la organización en los productos. Un activo es un artefacto o producto de trabajo desarrollado para ser utilizado en la producción de más de un producto en una línea de productos. Según [Clements 01] los activos núcleo, es decir, los activos que forman la base de la línea de productos software pueden ser la arquitectura, los componentes software reutilizables, modelos de dominio, especificación de requisitos, documentación y especificaciones, modelos de rendimiento, calendarios, presupuestos, planes de test, casos de test, planes de trabajo, descripciones de proceso, o cualquier otro resultado útil de construir un sistema.

Uno de los activos clave es la arquitectura software, es la llave para la reutilización sistemática, ya que describe la estructura de los productos del dominio, mostrando sus componentes y las relaciones entre los mismos. La arquitectura es la clave de la calidad y longevidad del producto y, por lo tanto, de la línea de productos.

"Una **arquitectura software** es la estructura o las estructuras que abarcan los componentes de software, las características externamente visibles de esos componentes y las relaciones entre ellas" [Bass 98]. En el caso de las líneas de productos la arquitectura software es uno de los activos clave. "Una **arquitectura de línea de productos software** o **arquitectura referencia** es una arquitectura común para un conjunto de productos o sistemas relacionados desarrollados por una organización" [Bosch 00]. Los ADL (Architecture Description Language) son lenguajes formales desarrollados específicamente para representar y analizar arquitecturas software. Existen ADLs que proporcionan mecanismos para capturar arquitecturas de línea de productos: *Mae* [Roshandel 04], *Koala* [van Ommering 00] y *xADL 2.0* [Dashofy 02].

La arquitectura software de un producto determina en gran medida los atributos de calidad alcanzables en ese producto; por lo tanto, la evaluación de la arquitectura software permite realizar una evaluación temprana sobre si los atributos de calidad requeridos son alcanzables o no. La evaluación de la arquitectura software se define como "el examen sistemático de la extensión con

la que una arquitectura software cumple los requisitos" [Dolan 01].

2.2.2. Ingeniería de línea de productos software

Según [Clements 01] en el enfoque de líneas de productos se pueden distinguir tres grupos de actividades: Desarrollo de activos núcleo o ingeniería de dominio, desarrollo de productos o ingeniería de aplicación y gestión de la línea. [Bosch 00] introduce, además otro aspecto muy importante que son las actividades de evolución o mantenimiento de la línea de productos.

La ingeniería de dominio tiene como objetivo identificar la parte común entre los diferentes productos y la parte variable entre los mismos para crear la infraestructura y el soporte necesario para conseguir el mayor grado posible de reutilización. Entre sus actividades principales se encuentran analizar los sistemas y crear los activos reutilizables. Se conoce como la fase de desarrollar para reutilizar.

La ingeniería de aplicación tiene como objetivo desarrollar aplicaciones o productos usando los activos creados en la ingeniería de dominio. Se conoce como la fase de desarrollar reutilizando. Es durante esta fase cuando los productos se construyen mediante la derivación de productos: "el proceso completo de construir un producto a partir de los activos software de la línea de productos [Deelstra 05]".

Las actividades de gestión de la línea de productos tanto a nivel técnico (proyecto) como organizativo (empresa) deben asegurar que el enfoque de línea de productos se siga adecuadamente y con éxito. Entre otras actividades la gestión supervisa el desarrollo de activos núcleo y el desarrollo de productos, es la encargada de la estructura organizativa, de asegurar los recursos necesarios, de crear un plan de adopción del enfoque de línea de productos, de asegurar que los objetivos y medidas son los adecuados, del feedback, de la elección del enfoque de la línea, etc.

Las actividades de evolución de la línea de productos se encargan de asegurar que la línea de productos se adapta adecuadamente a los nuevos requisitos tanto de los productos existentes como de nuevos productos que se tienen que incluir en la línea.

2.2.3. Variabilidad y técnicas de modelado

La variabilidad en un dominio distingue los miembros de una familia unos de otros [Clauß 01], es la habilidad de cambiar o personalizar un sistema [Gurp 01]

La variabilidad es un aspecto clave en las líneas de productos y, para poder gestionarla, varias técnicas de modelado de variabilidad han sido desarrolladas.

Varias de los enfoques se basan en el modelado de características que es uno de los modelos más utilizados para modelar la variabilidad en las líneas de productos: como FODA (Feature Oriented Domain Analysis) [Kang 90], FORM (*Feature Oriented Reuse Method*) [Kang 98], FOPLE (*Feature Oriented Product Line Software Engineering*) [Kang 02], FeatuRSEB [Griss 98], *Generative Programming* [Czarnecki 00], el enfoque de *Gurp et al* [Gurp 01], el enfoque de *Riebisch* [Riebisch 03], CBFM (*Cardinality based Feature Modelling*) [Czarnecki 04] y Forfamel [Asikainen 06]. Hay enfoques que utilizan casos de uso: [Halmans 03] o enfoques que utilizan modelos de características y casos de uso como PLUSS (*Product Line Use case modelling for Systems and Software engineering*) [Eriksson 05] y RequiLine [von˜der Maßen 03]. Otros enfoques modelan los puntos de variación como [Gomaa 04b], OVM (*Orthogonal Variability Modelling*) [Pohl 05] y VSL (Variability Specification Language) [Becker 03]. También hay enfoques que integran la variabilidad en los ADLs de línea de productos software Koala [van Ommering 00] y xADL [Dashofy 02]: Koalish [Asikainen 03] y [van˜der Hoek 04] respectivamente. Otras técnicas son Pure::Variants basado en el enfoque CONSUL [Beuche 04], ConIPF [Hotz 06] y GEARS [Krueger 02].

Varias de las técnicas utilizan UML (Unified Modelling Language) que es un estándar de facto en la industria para la notación de modelados software. Se proponen extensiones de UML para introducir la variabilidad como por ejemplo [Gomaa 04a] y [Clauß 01].

Respecto a los mecanismos para implementar la variabilidad y sus respectivos enfoques se pueden mencionar la programación generativa (*Generative prograraming*) [Czarnecki 00], la programación de marcos (*frames programing*) como XVCL [Zhang 04] y la programación orientada a características o FOP (*Feature Oriented Programming*): [Prehofer 97] y AHEAD [Batory 04].

2.2.4. Enfoques y herramientas existentes

Muchas de las técnicas y enfoques de modelado e implementación de la variabilidad se pueden considerar enfoques o métodos de desarrollo de líneas de productos; ya que tienen asociado un proceso para el desarrollo de la línea de productos software.

Como por ejemplo los enfoques basados en el modelado de características: FODA, FORM, FeatuRSEB, FOPLE, PLUSS, etc. El enfoque que propone [Pohl 05] que incluye el modelado OVM y [Gomaa 04a] que propone un proceso basado en UML para el desarrollo de líneas de productos software. Otros enfoques para desarrollo de línea de produtos son: FAST (*Family Oriented Abstraction Specification and Translation*) [Weiss 99], PuLSE (*ProdUct Line Software Engineering*) [Bayer 99], *Product Line Practice Initiative* del SEI [Clements 01], KobrA [Atkinson 02], QADA (*Quality-driven Architecture Design and quality Analysis*) [Matinlassi 02].

Respecto a las herramientas, existen varias para el modelado de características CaptainFeature [Bednasch], XFeature [Cechticky 04], FeaturePlugin [Antkiewicz 04], ASADAL/FORM que da soporte al proceso FORM [POSTECH 08], GuiDSL [Batory 05] (de AHEAD *tool suite*), la herramienta para el método RequiLine [von˜der Maßen 03] y FAMA [Benavides 07] para el análisis automático de los modelos de características.

Otras herramientas que soportan todo el proceso de desarrollo son: AHEAD Tool Suite [Batory 08], la herramienta GEARS [Krueger 02], pure::variants que soporta el proceso CONSUL [PureSystems 08] y la herramienta XVCL [Zhang 04].

2.2.5. Paradigmas relacionados

Relacionados a las líneas de productos software existen otros paradigmas y técnicas de desarrollo software como por ejemplo CBSE (*Component-based software engineering*), AOSD (*Aspect-Oriented Software Development*), FOP (*Feature Oriented Programming*) y MDD (*Model Driven Development*).

Algunos de estas técnicas ya se han mencionado como formas de implementar la variabilidad. Realmente estos paradigmas son adecuados para utilizarlas junto a la ingeniería de línea de productos software y la unión de estos para-

digmas con la ingeniería de línea de productos software ha dado paso a nuevos paradigmas.

CBSE (*Component-based software engineering*) o desarrollo basado en componentes es un paradigma que también ayuda a lograr la reutilización sistemática mediante el énfasis en la descomposición de sistemas en componentes lógicos con interfaces bien definidos. A la hora de desarrollar una línea de productos, el uso de componentes reutilizables es una opción muy utilizada un ejemplo de ello es Kobra [Atkinson 02] un método de ingeniería de línea de productos basada en componentes (*Component-based product line engineering*).

FOP (*Feature Oriented Programming*) es el estudio de la modularidad de las características y su uso en el síntesis de programas. AHEAD [Batory 04] es un ejemplo de enfoque FOP para líneas de productos.

AOSD (*Aspect-Oriented Software Development*) es una tecnología cuyo objetivo es ofrecer mecanismos para modularizar propiedades transversales (*crosscutting concerns*). Propone el uso de descripciones modulares de las propiedades transversales a través de una unidad modular nueva llamada aspecto. La variabilidad es a menudo de naturaleza transversal por lo que las técnicas AOSD son adecuadas para modularizar la variabilidad dando lugar al paradigma de desarrollo *aspect-based product line engineering*.

MDD (*Model Driven Development*) o desarrollo dirigido por modelos consiste en el uso sistemático de modelos como entidades de primera clase durante el ciclo de vida de ingeniería. La idea de unir MDD y las líneas de productos recibe el nombre de *Model Driven Software Product Lines* [Czarnecki 05a].

Hay también otros trabajos que combinan varios paradigmas como por ejemplo [Trujillo 07] que propone *Feature Oriented Model Driven Product Lines*, una unión de FOP, MDD e ingeniería de línea de productos. [Lopez-Herrejon 06] también combina varios paradigmas: AOSD, FOP y la ingeniería de línea de productos,

2.3. Atributos de calidad

La calidad en los sistemas es un tema considerado importante desde comienzos de la ingeniería del software cuando ya se hablaba de atributos tales como flexibilidad, integridad, rendimiento, mantenibilidad, etc. “La calidad software

es el grado en el que el software posee una combinación de atributos deseada" [IEEE 93]. Y un atributo de calidad es "una propiedad de un producto de trabajo o productos por los cuales su calidad es juzgada por algún *stakeholder*" [SEI 07c].

Los atributos de calidad también son conocidos como requisitos no-funcionales o extra-funcionales. A lo largo de la tesis se han evitado los términos no-funcional en la medida de lo posible porque se consideran términos que pueden confundir, ya que por requisito no-funcional se da a entender un requisito que puede ser especificado independientemente del comportamiento funcional del sistema [Bass 98]. Y muchos atributos de calidad como el rendimiento o la fiabilidad están muy unidos al comportamiento funcional.

La calidad software debe ser considerada en todas las fases de diseño, implementación y despliegue. Pero los diferentes atributos de calidad se manifiestan de forma diferente durante estas fases. Pero en general el diseño y la arquitectura software es crítica para la realización de muchos atributos de calidad, y estos atributos deben ser diseñados y evaluados a nivel de arquitectura [Bass 98].

Existen diferentes clasificaciones de atributos de calidad. Uno de los más utilizados es el de [Bass 98] que los divide en observables vía ejecución: rendimiento, seguridad, disponibilidad, funcionalidad, usabilidad, etc. y no observables vía ejecución: modificabilidad, portabilidad, reusabilidad, integrabilidad, testabilidad, etc. Clasificaciones similares son las que realizan [Bosch 00] que clasifica los atributos en atributos de calidad de desarrollo que son calidades del sistema relevantes desde la perspectiva de la ingeniería del software y operacionales que son calidades del sistema en operación, [Dolan 01] que divide los atributos en estratégicos porque tratan aspectos de negocio y de ciclo de vida y operacionales porque son importantes para la normal operación del sistema; y [Kazman 96] que los clasifica en tres categorías: atributos que se pueden describir observando la salida del sistema ejecutándose a partir de una entrada, atributos que se pueden describir midiendo las actividades de un equipo de desarrollo o mantenimiento y atributos que se pueden describir midiendo las actividades de un usuario particular (posiblemente otro sistema) en relación con el sistema ejecutándose.

Por lo tanto, existe consenso a la hora de clasificar los atributos de calidad en dos grandes categorías:

- Atributos operacionales o de ejecución
- Atributos de desarrollo o no observables vía ejecución

ISO 9126, un estándar internacional para la evaluación del software

[ISO/IEC 01], clasifica la calidad software en un conjunto estructurado de seis categorías de características (funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad) y subcategorías en cada una de ellas.

A pesar de la existencia de este estándar que define los atributos de calidad y otras definiciones disponibles [IEEE 90], en la práctica según el dominio donde se apliquen los atributos de calidad pueden tener diferentes acepciones por lo que hay múltiples definiciones. Además, a veces tienen significados imprecisos [Clements 02]. Esto hace que resulte muy difícil trabajar con atributos de calidad debido a su subjetividad y a que suelen estar pobremente definidos [Chung 99b].

2.4. Aseguramiento de calidad en líneas de productos software

Sin estrategias y técnicas adecuadas, el aseguramiento de la calidad no puede seguir el ritmo de ganancias de productividad del desarrollo y se convierte en el cuello de botella de la ingeniería de línea de productos. Como resultado, el objetivo de una reacción rápida y efectiva en costes a las necesidades del mercado y clientes es difícil de obtener. La importancia del aseguramiento de la calidad en las líneas de productos software y la necesidad de técnicas y métodos efectivos y eficientes para asegurar la calidad en el contexto de la ingeniería de línea de productos ha sido reconocida recientemente por la comunidad de las líneas de productos software. Hoy en día, cada vez más investigadores están investigando en formas de mejorar las técnicas y procesos de aseguramiento de calidad, así como desarrollando técnicas concretas para asegurar la calidad de los componentes reutilizables y de los productos de la línea de productos [Kolb 05].

En los últimos años, el interés por la calidad en contextos de reutilización ha crecido considerablemente. Se han organizado varios talleres: *eWorkshop on Quality Assurance for Software Product Lines: Strategic Issues* [Kolb 05], *International Workshop on Quality Assurance in Reuse Contexts* (QUARC'04) [Kolb 04], *The First IEEE International Workshop on Quality Oriented Reuse of Software* (QUORS'07) [COM 07].

Si en sistemas únicos lograr cumplir los atributos de calidad es a menudo un reto, en las líneas de productos software este reto se complica por la variabilidad en los atributos de calidad y las diferentes restricciones de calidad requeridas.

El análisis *trade-off* de los atributos de calidad es también más difícil que en sistemas únicos debido a la variabilidad y al número de posibilidades que crece exponencialmente. La dificultad crece pero también lo hace el impacto de no considerar los atributos de calidad. Las consecuencias de no considerar y gestionar la variabilidad en los atributos de calidad no son triviales. Si una línea de productos es desarrollada sin considerar la variabilidad en los requisitos de calidad, esta línea de productos no cubrirá todos los productos del alcance de la línea ni futuros nuevos productos. Como consecuencia, la inversión realizada al desarrollar la línea de productos no se recuperará.

Al ser la evaluación de la calidad en líneas de productos software el tema central de la tesis, se ha realizado un análisis de los retos asociados y de los métodos existentes que se presenta en el siguiente capítulo.

2.5. Conclusiones

El objetivo de este capítulo era proporcionar una introducción breve de los conceptos en el que basa la tesis: la ingeniería de línea de productos, los atributos de calidad y la evaluación de la calidad en el contexto de línea de productos software.

En el siguiente capítulo, se resume el análisis que se ha realizado de la evaluación de la calidad en las líneas de productos software, así como de las técnicas de modelado de la variabilidad en los atributos de calidad.

Capítulo 3

Modelado y evaluación de la calidad en las líneas de productos

3.1. Introducción

Este capítulo realiza un análisis de las técnicas y enfoques existentes para el modelado y la evaluación de los atributos de calidad en las líneas de productos software. Primeramente propone una nueva clasificación de atributos de calidad adaptada a las líneas de productos software. A continuación analiza la variabilidad en los atributos de calidad y las técnicas y enfoques existentes para su modelización. Por último, analiza la evaluación de atributos de calidad en líneas de productos software.

3.2. Atributos de calidad en las líneas de productos

Existen diversas clasificaciones y taxonomías de atributos de calidad: atributos de calidad de desarrollo y operacionales [Bosch 00], atributos estratégicos (los no observables vía ejecución) y operacionales (los observables vía ejecución), [Dolan 01]. ISO también [ISO/IEC 01] define un modelo de calidad del software con seis categorías de características (funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad) y subcategorías en cada una de ellas.

En el caso de las líneas de producto proponemos otra clasificación específica con dos tipos de atributos: Atributos de calidad específicos de línea de productos y atributos de calidad relevantes del dominio (ver figura 3.1). Los atributos específicos de línea de productos están relacionados con la flexibilidad que los activos software de la línea de productos deben poseer y los atributos relevantes de dominio están relacionados con el dominio específico de la línea de productos.

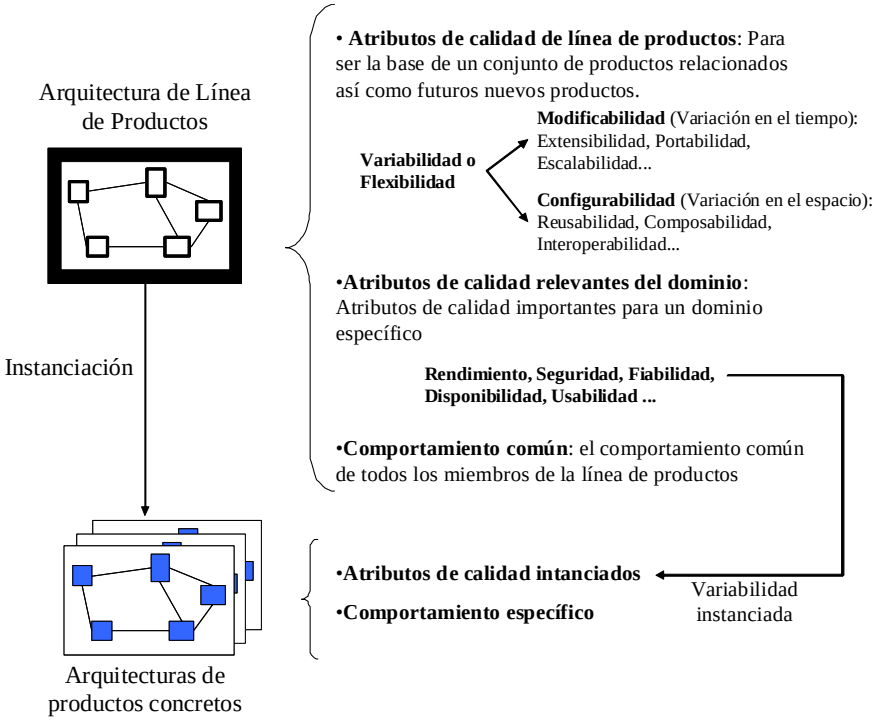


Figura 3.1: Clasificación de los requisitos de línea de productos

Los atributos de calidad de línea de productos son aquellos inherentes o específicos de las líneas de productos para permitir que la arquitectura sea la base para un conjunto de productos relacionados así como para futuros nuevos productos. Estos atributos están relacionados con la variabilidad o flexibilidad. Evaluando la variabilidad de una ALP (Arquitectura de Línea de Productos), se asegura que utilizando esa ALP es posible lograr toda la funcionalidad de todos los productos dentro del alcance definido para la línea. La variabilidad [Thiel 02] entendida como modificabilidad (para permitir la variación o evolución en el tiempo) y configurabilidad (variabilidad en el espacio) para obtener un conjunto de productos relacionados. Estos atributos de calidad suelen ser de desarrollo (o no observables vía ejecución).

Los atributos de calidad relevantes de dominio (tales como la seguridad

en dominios donde la seguridad es crítica, rendimiento en dominios de tiempo real, fiabilidad en sistemas empotrados, etc.) deben ser tenidos en cuenta en la ALP sino las implicaciones o consecuencias pueden ser muy serias y difíciles de corregir. Estos atributos usualmente son operacionales (observables vía ejecución). Como diferentes productos en el dominio pueden requerir diferentes valores en los atributos (no todos los productos requieren el mismo nivel de seguridad), la variabilidad en la manera en que el atributo se traslada al producto es relevante durante la evaluación para asegurar que se pueden obtener todos los atributos de calidad para todos los productos definidos en el alcance de la línea a partir de la ALP.

3.2.1. Variabilidad en los atributos de calidad

En una línea de productos, los miembros de la línea pueden requerir diferentes niveles de un atributo de calidad, por ejemplo pueden diferenciarse en términos de disponibilidad, seguridad (*security*), fiabilidad, etc. Un producto puede requerir una fiabilidad muy alta mientras que para otro, la fiabilidad no es importante. Incluso puede llegar a haber productos con la misma funcionalidad pero que difieren en niveles de atributos de calidad. La idea de variaciones en calidad se ha considerado en diferentes trabajos [Myllärniemi 06] [Halmans 03] [Niemelä 05a] y existen enfoques que se encargan de modelar la variabilidad teniendo en cuenta también la variabilidad en los atributos de calidad. Sin embargo, "hay una falta de comprensión total de los enfoques existentes para integrar la variabilidad en los atributos de calidad como parte de la gestión sistemática de la variabilidad de las líneas de productos software" como afirma [Myllärniemi 06]. La variabilidad de calidad se puede clasificar en diferentes tipos, Niemelä [Niemelä 05a] define los siguientes tipos:

- *Variabilidad de los atributos de calidad (opcionalidad)*: Por ejemplo, para un miembro de la línea la fiabilidad es importante pero para otros miembros no hay requisitos de fiabilidad.
- *Diferentes niveles de prioridad de los atributos:* Por ejemplo, para un miembro de la familia los requisitos de extensibilidad son extremadamente altos, mientras que para otros esos requisitos están al nivel más bajo.
- *Variación indirecta:* La variabilidad funcional puede indirectamente causar variación en los requisitos de calidad.

3.3. Modelado de la variabilidad en atributos de calidad

A pesar de que los atributos de calidad son un aspecto importante para el desarrollo de productos, han recibido muy poca atención en las técnicas de modelado de la variabilidad [Sinnema 07]. La mayoría de los métodos mencionados en el estado del arte ignora los atributos o sólo los menciona brevemente. Una excepción son CBFM y ConIPF que mencionan como modelar los aspectos de calidad. Sin embargo, no proporcionan una explicación detallada ni ningún ejemplo.

Como se ha mencionado, hay enfoques para especificar los requisitos de atributos de calidad que no consideran la variabilidad explícitamente y de la misma manera, hay enfoques para especificar requisitos variantes que no consideran los atributos de calidad [Myllärniemi 06].

El modelado de la variabilidad en los atributos de calidad en importante tanto para la evaluación como para una derivación correcta, teniendo en cuenta los aspectos de calidad.

Existen algunos trabajos y extensiones de las técnicas de modelado mencionadas que si consideran los atributos de calidad. A continuación se detallan estos enfoques que permiten modelar la variabilidad en los atributos de calidad.

3.3.1. *Comparativa de notaciones para describir la variabilidad en los atributos de calidad:*

Goal-based model [González-Baixauli 04]: Este enfoque propone utilizar el análisis orientado a metas (*goals*) [Chung 99b] en líneas de producto. La ingeniería de requisitos orientada a metas es un enfoque que se encarga de los atributos de calidad o requisitos no funcionales en sistemas de un sólo producto. Se proponen dos sub-modelos: un modelo de metas funcional y un modelo de *softgoal*. Los atributos de calidad se representan como *softgoals* y el funcionamiento de estos atributos se codifica en el sub-modelo de metas o *goals* funcionales como tareas. Se priorizan los *softgoal* para realizar el análisis. Se utilizan correlaciones para representar los enlaces entre metas funcionales y *softgoals*. Los enlaces de correlación tienen diferentes etiquetas de influencia (- -,-,?,+,++). Estas etiquetas cualitativas se convierten en valores cuantitati-

vos: un valor para la posibilidad de satisfacción y otra para la posibilidad de negación.

F-SIG (*Feature-softgoal interdependency graph*) [Jarzabek 06]: El objetivo de este enfoque es proporcionar un *framework* para registrar el razonamiento de diseño en forma de interdependencias entre características y atributos de calidad. Se propone un nuevo modelo que es la unión de un modelo de características y un SIG (Softgoal interdependency graph) [Chung 99b]. En F-SIG, las contribuciones explícitas e implícitas desde las características hasta los atributos de calidad se modelan. Para expresar el grado de influencia las correlaciones pueden tener una etiqueta: (*break*:- -, *hurt*:-, *unkown*: ?, *Help*: +, *Make*: ++).

COVAMOF [Sinnema 04]: COVAMOF es un *framework* de modelado de la variabilidad en familias de productos software. Con este *framework* es posible modelar la variabilidad en todas las capas de abstracción. Utiliza el CVV (*COVAMOF Variability View*) que propone dos vistas: La vista de puntos de variación y la vista de dependencias. Los atributos de calidad pueden ser modelados con dependencias, una dependencia puede ser una propiedad que expresa el valor de un atributo de calidad como rendimiento o uso de memoria. Y la asociación se utiliza para relacionar los puntos de variación y las dependencias.

Extended feature model [Benavides 05]: Propone una extensión del modelo de características para tratar con características extra-funcionales. Extiende la notación con atributos, propiedades de una característica que pueden ser medidas como por ejemplo disponibilidad, coste, latencia, ancho de banda y las relaciones entre estos atributos. Cada característica puede tener una o varias relaciones de atributos que toman un rango de valores de un dominio discreto o continuo. También proporciona razonamiento automático usando CSP (*Constraint Satisfaction Problems*).

Definition hierarchy [Kuusela 00]: Este método en un árbol *AND* lógico donde los nodos de arriba son objetivos de diseño: *drivers* arquitecturales y otros atributos de calidad que el sistema debe cumplir. Los otros nodos son decisiones de diseño y las aristas entre un objetivo de diseño y una decisión de diseño muestran que ese requisito es satisfecho (parcialmente) por las decisiones de diseño. Cada nodo en el árbol toma una prioridad que refleja la importancia de ese nodo para conseguir la intención de su padre. Las prioridades son específicas de producto; se utilizan como mecanismo para describir productos.

El enfoque QoS [Capilla 02]: Este enfoque propone extender FODA para definir las propiedades y parámetros de calidad de servicio o *quality-of-serive* (QoS) de los nodos de una arquitectura distribuida. Los atributos QoS se representan "como características QoS" mediante etiquetas y se especifican los

valores entre corchetes cerca del nombre de la característica. También añaden semántica extra si es necesario para representar valores máximos o mínimos, rangos, número de veces que se instancia, acción a ejecutar si se cumple una condición, etc.

Bayesian Belief Network: Zhang et al. [Zhang 03] proponen un enfoque basado en una red bayesiana o *Bayesian Belief Network* (BBN) para predecir y asegurar la calidad en una línea de productos. El BBN es usado para modelar explícitamente el impacto de las variantes (especialmente las decisiones de diseño) en los atributos de calidad del sistema. El modelo de características se utiliza para capturar los requisitos funcionales y el modelo BBN para capturar el impacto de las variantes funcionales en los atributos de calidad.

El método *Quality Requirements of a Software Family* (QRF) [Niemelä 05b] captura y mapea los requisitos de una familia de productos a la arquitectura de la familia analizando para ello las necesidades de los participantes de negocio y desarrollo y el impacto de estas necesidades en la arquitectura. Consiste en cinco pasos: *Análisis del impacto* en el que se utiliza el *framework* i* [Chung 99a]: un modelo de dependencias estratégicos que define los requisitos de los participantes tanto funcionales como de calidad; *análisis de la calidad* donde los requisitos se priorizan (bajo, medio, alto) y se expresan de forma que más tarde puedan ser trazados y medidos; *análisis de la variabilidad* donde los requisitos de calidad que varían en el dominio de negocio son definidos utilizando el modelo de dependencias estratégicos; *análisis del dominio jerárquico* para mapear los atributos de calidad comunes y variables a la funcionalidad y *representación de la calidad* para describir la arquitectura de forma que los requisitos de calidad pueden ser evaluados utilizando los modelos arquitectónicos. Para este último paso se utilizan los estilos y patrones arquitectónicos y un *framework* NFR (*Non-Functional Requirements*) para realizar el análisis *trade-off* y seleccionar el estilo que mejor cumple los requisitos de calidad y los perfiles para extender los modelos arquitectónicos para soportar ciertos aspectos de calidad.

Los enfoques mencionados han sido analizados para ver si consideran todos los requisitos que consideramos importantes para modelar la variabilidad en los atributos de calidad:

- *Razonamiento automático:* Diferentes tareas de razonamiento pueden interesar: conseguir un valor aproximado o nivel para un atributo de calidad o varios partiendo de un conjunto de requisitos funcionales, detectar configuraciones imposibles a partir de un conjunto de requisitos funcionales y niveles de calidad, detectar conflictos entre atributos de calidad y proporcionar información para realizar un análisis de *trade-off*... Debi-

do a la complejidad de este análisis y del razonamiento, es aconsejable que se realice de forma automática. Para lograr que sea automático se requiere de técnicas de inteligencia artificial tales como *Constraint Satisfaction Problems* (CSP), *Boolean Satisfiability Problems* (SAT) y *Binary Decision Diagrams* (BDD) [Benavides 06].

- *Caracterización de los atributos de calidad:* Los atributos de calidad tienen definiciones vagas. En diferentes dominios pueden llegar a tener significados muy diferentes. Por lo tanto, es necesario concretizar y hacer que los atributos de calidad sean específicos. Un mecanismo para describir y explicar los atributos de calidad de forma adecuada es necesario: una estructura donde los atributos de calidad pueden ser explicados mediante refinamientos.
- *Opcionalidad*: En un producto un atributo puede ser importante pero en otro no estar requerido. Debe haber mecanismos para representar esta variabilidad a nivel de línea de productos.
- *Niveles:* Se necesitan diferentes niveles de prioridad en los atributos dependiendo de los productos. Sin embargo, los atributos de calidad no son fáciles de cuantificar, formas de refinar y especificar los atributos de forma medible son necesarios así como la forma de definir niveles (alto, medio, bajo).
- *Impactos cuantitativos y cualitativos*: La variación indirecta debe representarse mediante impactos que pueden ser cualitativos o cuantitativos. Además, se debe proporcionar formas de cuantificar los impactos para permitir un análisis automático.
- *Impactos de grupo*: Hay relaciones de impacto de un grupo de características que si interaccionan no se pueden definir de forma individual. Es decir, el impacto de dos variantes a la vez no es siempre la suma del impacto de estas variantes por separado.

Ninguna de las propuestas evaluadas cumple todos los requisitos identificados (ver tabla 3.1).

Tabla 3.1: Enfoques para representar variabilidad en los atributos de calidad

	Razonamiento automático	Caracterización de los atributos	Opcionalidad a nivel de LP	Niveles de prioridad	Cuantitativo y cualitativo	Impactos de grupos (interacciones)
Goal-based model	Sí	Sí	No	No	Sí	No
F-SIG	No	Sí	No	No	No	No
COVAMOF	Sí	No	No	No	Sí	Sí
Extended FM	Sí	No	No	No	No	No
QoS	No	No	Sí	No	Sí	No
Definition Hierarchy	No	Sí	+/-	No	Sí	No
BBN	Sí	No	No	+/-	Sí	Sí
QRF	No	+/-	Sí	Sí	No	No

3.4. Evaluación de atributos de calidad en líneas de productos

En esta sección, se analizan los métodos existentes así como los nuevos retos que surgen en el contexto de la evaluación de los atributos de calidad en las líneas de productos software. La evaluación de si un sistema cumple los atributos de calidad puede realizarse a diferentes niveles de abstracción: diseño (evaluación de la arquitectura), implementación (testeos), etc. Se analizan la evaluación de las arquitecturas software por el gran impacto que tienen en los atributos de calidad y la evaluación mediante testeo o mediciones, una forma de validación clásica de la calidad.

3.4.1. Evaluación de la arquitectura software

La arquitectura software de un sistema tiene una gran influencia en la calidad final de los sistemas ya que puede inhibir o permitir los atributos de calidad de los productos. Teniendo en cuenta esto, la evaluación de la arquitectura puede ser muy útil. Analizar el potencial de una arquitectura para obtener los niveles de calidad requeridos ayuda a encontrar problemas de una forma temprana en el ciclo de vida, cuando son más sencillos y baratos de corregir que en etapas posteriores como la implementación, el testeo o el despliegue. Además, la evaluación de la arquitectura software ayuda a mejorar la comunicación de los agentes (*stakeholders*), mejorar la documentación y priorizar los objetivos de calidad.

La evaluación de una arquitectura se define como "el examen sistemático de la extensión con la que una arquitectura software cumple los requisitos" [Dolan 01]. Estos requisitos pueden ser tanto funcionales como atributos de calidad. La mayoría de métodos de evaluación se centran en los atributos de calidad, ya que la calidad es algo que debe ser considerada en el diseño y no se puede añadir más tarde. De ahí la importancia de evaluar los atributos de calidad en la arquitectura software.

En el caso de las arquitecturas de línea de productos (ALP) la evaluación de la arquitectura es crucial para asegurar que la ALP es lo suficientemente flexible para soportar los diferentes productos y permitir la evolución. Otra de las características específicas de las líneas de productos es que hay dos niveles de abstracción arquitectónica donde se puede realizar la evaluación (la

arquitectura de línea de productos software y las arquitecturas de los productos derivados). En el caso de los productos derivados de la línea de productos, evaluar todos ellos puede resultar muy costoso y en la medida de lo posible hay que intentar acortar las evaluaciones de la arquitectura de los productos, evaluando todo lo posible a nivel de línea de productos.

También hay factores organizacionales que afectan a las evaluaciones de arquitectura de línea de productos: una ALP involucra a más *stakeholders* que una arquitectura de un único sistema porque el alcance es mayor. Además, los *stakeholders* pueden ser de diferentes departamentos en diferentes ciudades e incluso diferentes países.

La aplicación clásica de la evaluación de la arquitectura ocurre cuando la arquitectura software o arquitectura de línea de productos se ha especificado y antes de comenzar a implementar. Sin embargo, la evaluación de la arquitectura puede aplicarse en cualquier momento de la vida de la arquitectura y particularmente en el contexto de las líneas de productos surgen nuevos momentos de evaluación.

Todos estos aspectos deben ser considerados cuando se evalúa una ALP y por lo tanto, las técnicas y enfoques de evaluación existentes tienen que hacer frente a nuevos retos.

A continuación, se describen los momentos en los que se puede realizar una evaluación en una línea de productos.

3.4.1.1. Momentos de evaluación en una línea de productos

En desarrollos tradicionales, la evaluación de la arquitectura se realiza normalmente durante el diseño. En el contexto de una línea de productos, la evaluación de la arquitectura puede ser útil en diferentes momentos (ver figura 3.2).

Normalmente la evaluación de la arquitectura se realiza durante el diseño de la arquitectura. En las líneas de productos software, esta evaluación se replica en la ingeniería de dominio y en la ingeniería de aplicación. Durante la ingeniería de dominio, la evaluación (**evaluación de la ALP**) asegura que la arquitectura referencia cumple los atributos de calidad de la línea de productos, los atributos de calidad relevantes de dominio y el comportamiento común. Puede ser muy útil para detectar aspectos problemáticos y puntos de riesgo o para

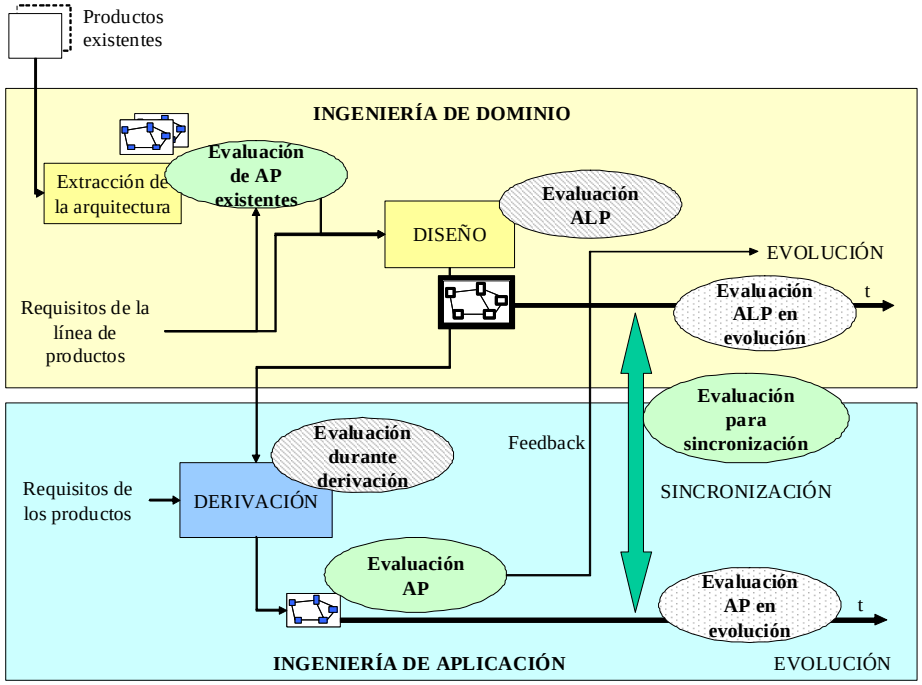


Figura 3.2: Momentos de evaluación en el contexto de una línea de productos

comparar arquitecturas candidatas y seleccionar la que mejor soporta los atributos de calidad requeridos. Durante la ingeniería de aplicación (**evaluación de arquitectura de producto (AP)**), la evaluación asegura que los atributos de calidad y el comportamiento específico de producto se cumplen, así como la conformidad de la arquitectura respecto a la arquitectura referencia.

Durante la evolución, puede ser necesario o deseable adaptar la arquitectura referencia para incluir nuevos requisitos (debido a nuevos productos o nuevos requisitos de producto). La evaluación ayuda a analizar la magnitud de los cambios arquitecturales requeridos (**Evaluación de la ALP en evolución**) y de este modo a decidir si los requisitos se van a considerar a nivel de línea de productos. También durante la evolución, las ALPs y APs evolucionan a partir del diseño inicial y los cambios pueden causar que los atributos de calidad ya no se cumplan. La evaluación en este caso, es útil para asegurar que la arquitectura sigue cumpliendo su objetivos de calidad (**Evaluación de la ALP en evolución** y **evaluación de la AP en evolución**). La evaluación de la arquitectura en evolución está muy relacionada con la recuperación de la arquitectura porque algunas veces ya no se dispone de una descripción actualizada de la arquitectura.

Hay tres nuevos momentos de evaluación específicos para las líneas de produc-

tos: antes de desarrollar la arquitectura referencia (**Evaluación de la arquitectura de un producto existente**) para analizar y comparar arquitecturas de productos existentes para usarlos como base para la línea de productos. Durante la instanciación o derivación de productos (**Evaluación durante la derivación**) para comparar variantes alternativas que afectan a los atributos de calidad. La derivación de productos consiste en construir productos individuales utilizando los artefactos software reutilizables y durante este proceso es necesario tomar decisiones arquitecturales que pueden afectar a los atributos de calidad de los productos. La evaluación durante la derivación no es necesaria si se analizan los efectos de las variantes en los atributos de calidad durante la evaluación de la ALP. Y por último, durante la actualización de la arquitectura (**Evaluación para sincronización**) para identificar cómo las modificaciones y mantenimiento de la ALP afecta a los productos ya en el mercado y en el otro sentido, cómo los cambios en los productos afectan a la línea de productos con el objetivo de mantener la coherencia entre la ALP y las arquitecturas concretas (evaluar el impacto de requisitos nuevos en los productos, detectar variabilidad que ya no es necesaria, etc.). Este tipo de evaluación está muy relacionada con la evolución y la derivación.

No es rentable evaluar la arquitectura en cada uno de los momentos. El arquitecto software debe seleccionar los momentos más apropiados para evaluar dependiendo del caso. En líneas de productos de generación automática, la evolución de los productos puede no tener sentido y que todos los cambios se realicen a nivel de línea; en este caso, la evaluación para sincronización o la evaluación de la AP en evolución no serían aplicables. En una línea de productos ya desarrollada, la evaluación de ALP o la evaluación antes de desarrollar no son aplicables pero la evaluación de ALP en evolución puede ser interesante.

3.4.1.2. Métodos de evaluación de la arquitectura software

Existen diversos tipos de métodos de evaluación de arquitecturas software: métodos genéricos (útiles para más de un atributo) pero que se utilizan para un sólo atributo en cada evaluación, métodos multi-atributos para evaluar más de un atributo y realizar un análisis *trade-off* y métodos específicos para un atributo de calidad.

Algunos de los métodos de evaluación más conocidos son SAAM, ATAM [Clements 02] y SBAR [Bengtsson 98]. SAAM es genérico; puede utilizarse para varios atributos de calidad mientras que ATAM y SBAR son métodos multi-atributo. Existen también varios métodos para atributos de calidad espe-

cíficos: PASA [Williams 02] para rendimiento, SALUTA [Folmer 03] para usabilidad, ALMA [Bengtsson 04] para modificabilidad, ALPSM [Bengtsson 99] para mantenibilidad, etc. Sin embargo todos estos métodos están pensados para arquitecturas de un sólo producto. Se puede encontrar más información de estos métodos en el anexo E.

Una arquitectura de línea de productos debe soportar los requisitos compartidos por todos los sistemas de la línea, así como, los requisitos diferentes de cada uno de los sistemas individuales, de modo, que sea posible obtener todos los sistemas a partir de la arquitectura genérica o de línea de productos. Estos aspectos de las ALP deben ser considerados a la hora de evaluar y por lo tanto, plantean varios desafíos para las técnicas y enfoques de evaluación existentes [Etxeberria 05b].

Algunos de los métodos mencionados para evaluar arquitecturas de un sólo producto también pueden utilizarse en líneas de producto: SAAM, ATAM [Clements 02]. Estos métodos aunque no son específicos de línea de productos, son adecuados para evaluar atributos de calidad de desarrollo, o específicos de línea de productos tales como la mantenibilidad y la extensibilidad entre otros. ATAM ha sido utilizado en el contexto de líneas de productos [Ferber 02b] [Gallagher 00] pero sin ningún tratamiento especial, los aspectos particulares de las líneas de productos eran tratados implícitamente como requisitos de calidad.

Otro caso de aplicación de métodos para un sólo producto en líneas de productos software es el realizado en Nokia [Rosso 06]. A partir de su experiencia en la evaluación utilizando métodos existentes, han realizado pequeñas adaptaciones a estos métodos para aplicarlos en sus líneas de productos software. En concreto han puesto en práctica un método basado en escenarios para las evaluaciones que se basa en ATAM, SAAM y SBAR, un método para el rendimiento a nivel arquitectónico que se basa en [Smith 02] y han adaptado el método basado en la experiencia para la evaluación de arquitecturas que propone [Bosch 00].

Comparativa de métodos específicos de evaluación arquitectónica en el contexto de líneas de productos

También existen métodos específicos de evaluación arquitectónica en el contexto de líneas de productos. A continuación se clasifican estos métodos según lo que evalúan:

- Para evaluar arquitecturas de línea de productos en fase de diseño existen diferentes métodos, como: FAAM (*Family Architecture Assessment*

Method) [Dolan 01] para evaluar arquitecturas de familias de sistemas de información, los métodos RAP (*Reliability and Availability Prediction*) [Immonen 06] y el método IEE (*Integrability and Extensibility Evaluation*) [Henttonen 07] desarrollados dentro de la metodología QADA (*Quality-driven Architecture Design and quality Analysis*) [Matinlassi 02] para analizar arquitecturas de línea de productos, REDA (*Reliability Evaluation of Domain Architectures*) [Auerswald 02] para analizar la fiabilidad de una ALP y D-SAAM (*Distributed SAAM*) [Graaf 05], una variante de SAAM para evaluar arquitecturas referencia.

- Para evaluar arquitecturas de línea de productos existentes (evolución): *Gannod y Lutz* [Gannod 00] proponen un enfoque que evalúa requisitos de calidad y funcionales, *Maccari* [Maccari 02] propone un método para evaluar la evolución y *Riva y Rosso* [Riva 03] adaptan el enfoque de *Maccari*.
- También hay algunos métodos para evaluar la variabilidad, uno de los aspectos clave en las líneas de productos. A nivel arquitectónico: SBA (*Scenario-Based Architecting*) [America 05] es un método para identificar y cuantificar los beneficios potenciales de las diferentes opciones de variabilidad arquitectónica. Para todos los niveles de abstracción y no sólo a nivel arquitectónico existen el enfoque de *Wijnstra* [Wijnstra 03] y COSVAM (*The COVAMOF Software Variability Assessment Method*) [Deelstra 04].
- Existen también métodos orientados a evaluar arquitecturas de productos existentes para utilizarlos como base para las líneas de productos: SACAM (*Software Architecture Comparison Analysis Method*) [Stoermer 03] es un método para comparar arquitecturas y el enfoque de *Korhonen* [Korhonen 04] analiza si una arquitectura puede ser usado como base para una línea de productos.
- Para evaluar arquitecturas de productos derivados existe el método TPA (*Timing Property assessment*) [Alonso 98] que reutiliza modelos RMA (*Rate Monotonic Analysis*) de componentes individuales para derivar el modelo RMA global del sistema.
- El único método que se puede utilizar para evaluar tanto la arquitectura de línea de productos como las arquitecturas derivadas obtenidas a partir de la ALP es el método HoPLAA (*Holistic Product Line Architecture Assessment*) [Olumofin 05] [Olumofin 07] que es una adaptación de ATAM para las líneas de productos.
- También hay métricas específicas definidas para PLA: métricas de utilización de servicios [van˜der Hoek 03] y las métricas de *Rahman* [Rahman 04].

Tabla 3.2: Métodos de evaluación de arquitecturas de línea de productos

Método	Objetivo	Tipo de atributo	Técnica de evaluación	Descripción del proceso	Validación del método	Relación con otros métodos
FAAM (Family Architecture Assessment Method)	Evaluación orientada a los agentes (*stakeholders*) de arquitecturas de familias de sistemas de información	Atributos de calidad de LP: Interoperabilidad, extensibilidad...	Escenarios, otras técnicas	Muy detallado: Pasos, guías, roles ...	2 casos de estudio en diferentes dominios	Extiende SAAM
REDA (Reliability Evaluation of Domain Architectures)	Evaluación de ALP para predecir la fiabilidad	Atributos de calidad de dominio: Fiabilidad	Casos de fallo, modelo de fiabilidad cualitativo (QRM), métricas...	Razonable: Pasos, técnicas ...	Caso de estudio de sistemas de control automotor	-
D-SAAM (Distributed SAAM)	Evaluación de arquitecturas de referencia reduciendo el impacto organizacional	Atributos de calidad de LP: Mantenibilidad	Escenarios	Bien explicado: Pasos, guías, roles ...	Aplicado en una LP de sistemas copiadoras	Variante de SAAM
RAP (Reliability and Availability Prediction)	Método para evaluar la fiabilidad y la disponibilidad	Atributos de calidad de dominio: Fiabilidad y disponibilidad	Modelos de cadenas Markov, simulaciones, estimaciones	Bien explicado: Pasos, guías...	Un caso de ejemplo	-
IEE (Integrability and Extensibility Evaluation)	Método para evaluar la integrabilidad y la extensibilidad	Atributos de calidad de LP: Integrabilidad y extensibilidad	Escenarios	Razonable: Pasos, guías...	Un caso de ejemplo	-
Enfoque de Gannod y Lutz	Analizar una ALP existente	Atributos de calidad de LP: Modificabilidad	Escenarios y *model checking*	Razonable: Pasos, guías...	Aplicado en una LP de telescopios	Incluye un paso similar a SAAM
Enfoque de Maccari	Evaluar la capacidad de una ALP para adaptarse a la evolución	Atributos de calidad de LP: Relacionados con la evolución: escalabilidad, modificabilidad...	Escenarios	Brevemente explicado e ilustrado a través de casos de estudio	2 casos de estudio en diferentes dominios	-
Enfoque de Riva y Rosso	Evaluar las ALPs para la evolución	Atributos de calidad de LP: Flexibilidad, modificabilidad	Escenarios, análisis basado en la experiencia	Explicado con un caso de estudio	Un caso de estudio en una LP de terminales móviles	Adapta el enfoque de Maccari
SBA (Scenario-Based Architecting)	Identificar y cuantificar los beneficios de diferentes opciones de variabilidad	Atributos de calidad de LP: Variabilidad	Análisis cuantitativo basado en escenarios	Bien explicado: Pasos, guías,...	2 casos de estudio del dominio médico	Utiliza SQUASH [Svahnberg 03]

COSVAM (The COVAMOF Software Variability Assessment Method)	Evaluar la variabilidad de una LP en un contexto de evolución	Atributos de calidad de LP: Variabilidad	Escenarios de productos, análisis basado en expertos...	Bien explicado: Pasos, guías, roles ...	Aplicado en una LP de sistemas de tráfico inteligente	-
Enfoque de Wijnstra	Evaluar la forma en la que una LP lidia con la variación	Atributos de calidad de LP: Variabilidad	Estudio de la información capturada	Visión de conjunto	Caso de estudio en el dominio médico	-
SACAM (Software Architecture Comparison Analysis Method)	Comparar arquitecturas candidatas (arquitecturas de productos existentes)	Atributos de calidad de LP Atributos de calidad de dominio	Escenarios, tácticas y métricas	Explicación detallada: Pasos, guías, participantes...	Un ejemplo para ilustrar el método	-
Enfoque de Korhonen	Evaluar la adaptabilidad del sistema para convertirse en línea de productos	Atributos de calidad de LP: Adaptabilidad, configurabilidad... Atributos de calidad de dominio: Fiabilidad, rendimiento...	Escenarios	Explicado con un caso de estudio	Aplicado en un caso de estudio de máquinas móviles.	Aspectos similares a SAAM y ATAM
HoPLAA	Evaluación de arquitecturas de línea de productos en una forma integrada y holístico, tanto a nivel de línea como a nivel de productos	Atributos de calidad de LP Atributos de calidad de dominio	Escenarios	Explicado comprensiblemente	Un ejemplo	Adaptación de ATAM
TPA (Timing Property Assessment)	Analizar las propiedades de tiempo de nuevos productos de una línea	Atributos de calidad de dominio instanciados: propiedades de tiempo	RMA	Descripción general	Un pequeño ejemplo para ilustrar el método	-
Métricas de utilización de servicio	Evaluar y mejorar las ALPs	Atributos de calidad de LP: Solidez estructural	Métricas	Explicado comprensiblemente	Un caso de estudio en una LP biblioteca digital	-
Métricas de Rahman	Medir los atributos de calidad de una ALP	Atributos de calidad de LP: Reusabilidad, modularidad	Métricas	Razonable	Caso de estudio en un sistema biblioteca	Incluye métricas de utilización de servicio

3.4.2. Testeo de atributos de calidad en líneas de producto

"El testeo es un enfoque de validación y verificación de los artefactos producidos. Se refiere a cualquier actividad que valide y verifique a través de la comparación de un resultado actual con el resultado que se espera que el artefacto produzca, basándose en sus especificaciones. Las desviaciones respecto a los resultados esperados son llamados fallos (*failures*)". "Un fallo es considerado el resultado de un defecto en el artefacto" [McGregor 01]. El testeo de la calidad se utiliza para asegurar que el producto cumple los requisitos de calidad. En el caso de una línea de productos, esos requisitos de calidad pueden tener a su vez variabilidad.

Hay varios enfoques para testeo de líneas de producto software: *Nebut at al.* [Nebut 06] define un enfoque para convertir en automático la generación de test, el enfoque ScenTED (*Scenario-based Test case Derivation*) [Pohl 06] también facilita la derivación de casos de test desde los casos de uso, *McGregor* [McGregor 01] también presenta prácticas de testing para líneas de productos, la metodología de testeo PLUTO [Bertolino 03], herramientas de testo para líneas de producto como RITA [Kauppinen 03], etc. Sin embargo, la mayoría de enfoques se centra en el testeo de requisitos funcionales y los requisitos de calidad no se consideran explícitamente. Sólo se tiene constancia una técnica de testeo de líneas de productos que considere un atributo de calidad (*performance*) [Reis 06].

3.5. Conclusiones

Se percibe una diferencia entre la variabilidad funcional y la variabilidad en los atributos de calidad (presente en los atributos de calidad relevantes de dominio). Ciertamente, la variabilidad funcional ha recibido mucha más atención que la variabilidad en la calidad.

Las técnicas existentes para modelar la variabilidad no proporcionan la necesaria caracterización de los atributos teniendo en cuenta su ambigüedad ni la forma de representar los impactos de las variantes funcionales en los atributos de calidad respondiendo a todos los tipos de variabilidad en los atributos de calidad mencionados previamente. Otros enfoques que permiten modelar variabilidad en los atributos de calidad tampoco responden a todos los requisitos

definidos.

Respecto a los métodos de evaluación y testeo, la mayoría se centra en atributos de calidad de desarrollo o propios de la línea de productos tales como extensibilidad, mantenibilidad, etc. Los métodos para atributos de dominio son específicos para un atributo como por ejemplo RAP y no todos los atributos están cubiertos. Uno de los métodos más prometedores es HoPLAA, un método de evaluación multi-atributo que permite realizar análisis *trade-off* entre los atributos. Sin embargo, el soporte de HoPLAA para atributos de calidad operacionales es escaso.

Capítulo 4

Visión general del método CaLiPro (Evaluación de atributos de Calidad en Líneas de Productos)

4.1. Introducción

El presente capítulo introduce el método CaLiPro (Evaluación de atributos de **Ca**lidad en **Li**neas de **Pro**ductos software). Este método sirve para modelar y gestionar la variabilidad (funcional y de calidad) durante todo el ciclo de vida, con objeto de facilitar la validación de la calidad de la línea y permitir una derivación que tenga en cuenta los niveles de calidad de los atributos de calidad operacionales.

El método utiliza un modelo de características extendido que permite modelar la variabilidad en los atributos de calidad y se completa con un proceso para especificar dicho modelo y realizar evaluaciones de los atributos de calidad, así como para facilitar la derivación guiada por atributos de calidad.

La figura 4.1 muestra como se incluye el proceso propuesto en el ciclo de vida de la ingeniería de línea de productos software de [Czarnecki 98]. El proceso es transversal a actividades como el análisis, el diseño y la implementación por lo que se puede incluir en la mayoría de los ciclos de vida de línea de productos software.

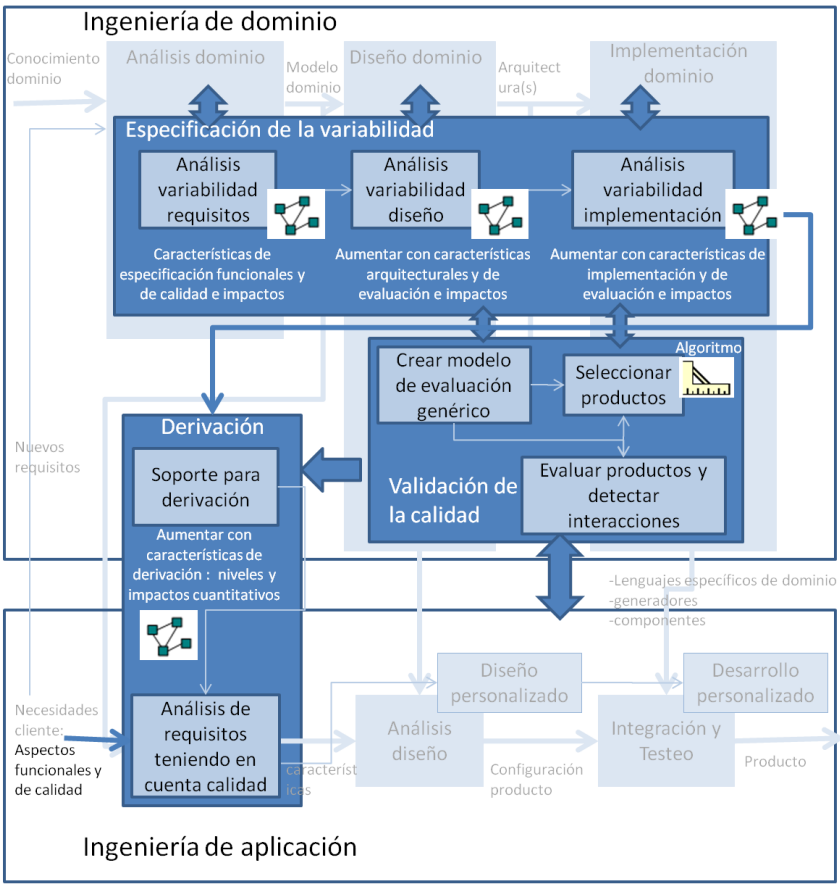


Figura 4.1: Fases del proceso CaLiPro

Durante la ingeniería de dominio se especifica el modelo de características extendido que facilita la validación de atributos y la derivación. Para la validación de los atributos operacionales de la línea es necesario generar un modelo genérico de evaluación y/o derivar diseños o productos para evaluarlos y extrapolar los resultados a toda la línea. El uso del modelo también se sitúa dentro de la ingeniería de aplicación, ya que en la especificación de los requisitos de los productos se cuenta con información de calidad incluida en el modelo que facilita la derivación. La definición completa del proceso se encuentra en el anexo C.

A continuación se explican las tres fases principales del proceso: especificar, validar y derivar:

- **Fase Especificación de la Variabilidad:** Esta fase construye un modelo de variabilidad (Modelo de características extendido) que recoge la variabilidad a diferentes niveles de abstracción: especificación de requisitos tanto funcionales como de calidad, decisiones de diseño y de implementación variantes y características de evaluación. El modelo permite relacionar las características funcionales con los atributos de calidad.
- **Fase Validación de la Calidad:** Esta fase testea y comprueba que la línea soporte los aspectos de calidad clave de los productos. Para que la validación se pueda hacer de manera rentable, se define un modelo de evaluación genérico (con variabilidad, válido para evaluar cualquier producto de la línea) y/o se seleccionan productos clave que permitan extrapolar los resultados de la evaluación a la línea (mediante un algoritmo que selecciona el mínimo número de productos para detectar interacciones entre las características y cuantificar los impactos para poder extrapolar los datos a todos los productos).
- **Fase Derivación:** Primeramente se aumenta el modelo de características extendido con características de derivación de los atributos de calidad. Posteriormente, los productos se derivan teniendo en cuenta la influencia de las características funcionales, arquitecturales y de implementación en la calidad final del producto.

En las siguientes secciones se detallan y razonan las características del método: el tipo de evaluación a la que está dirigida y el modelo que utiliza.

4.2. Evaluación de la calidad

El objetivo del método CaLiPro es facilitar la evaluación de la calidad en las líneas de productos software. La evaluación de la calidad definida como "el examen sistemático de la extensión con la que una entidad es capaz de cumplir los requisitos especificados" [ISO/IEC 99].

La calidad software se especifica mediante atributos de calidad como rendimiento, mantenibilidad, usabilidad, portabilidad, seguridad, etc. En nuestro caso, el método está orientado a evaluar los atributos de calidad de dominio que son los que presentan variabilidad en las líneas de productos software y para los que existe menos soporte de técnicas y herramientas adaptadas debido a la complejidad que añade la variabilidad. Por lo tanto, la evaluación se va a centrar en medir dichos atributos, que son atributos de calidad operacionales.

Dependiendo de la entidad o artefacto que se evalúe, la validación de la calidad se puede realizar en diferentes niveles de abstracción: requisitos, diseño, implementación, etc.

En el caso del método CaLiPro, la evaluación se realiza a nivel de diseño y/o de implementación.

Durante el diseño es cuando se toman las decisiones que más afectan a los atributos de calidad y por lo tanto, donde se localiza la variabilidad que más impacta en la calidad y su variabilidad. Además, la evaluación del diseño permite una validación temprana de los atributos antes de llegar a la implementación y cuando los errores son más fáciles de corregir. En concreto, la evaluación se realiza sobre la arquitectura software, el artefacto de diseño más representativo de un sistema.

A nivel de implementación, también hay decisiones que afectan a la calidad pero en menor medida. Sin embargo, un producto ya implementado puede ser ejecutado y esto permite medir los atributos de calidad que son operacionales u observables vía ejecución, como es el caso de los atributos de calidad de dominio. Al hablar de evaluación del código y su ejecución para medir ciertos aspectos, se podría decir que es un testeo orientado a medir atributos de calidad.

Además del término de evaluación, durante el documento también se utiliza el término validación. La validación es el proceso de asegurar que un producto o sistema cumple con los requisitos que tiene especificados. La evaluación de la arquitectura se puede considerar una forma de validación de artefactos, así

como el testeo software es una forma de validación del código [Clements 01].

A partir de un estudio de los métodos de evaluación de arquitecturas software existentes (ver capítulo 3 para más detalles) se ha concluido que los métodos de evaluación de arquitecturas de un solo sistema no soportan la variabilidad, mientras que la mayoría de métodos específicos de línea de productos se centra en atributos de calidad de desarrollo o específicos de la línea de productos tales como extensibilidad, modificabilidad, etc. Los pocos métodos que son multi-atributo o específicos para un atributo de calidad no proporcionan soporte para evaluar otros atributos de calidad operacionales.

En cuanto a los métodos de testeo o medición de atributos de calidad, la situación es todavía peor. El testeo funcional en las líneas de productos software ha recibido mucha atención, pero el testeo de requisitos de calidad apenas.

Sin embargo, el testeo o medición de atributos de calidad y la evaluación de arquitecturas software de un solo sistema son campos maduros donde existen muchos métodos específicos para atributos de calidad operacionales. Extender estos métodos para que soporten la variabilidad puede ser una opción, pero sería inviable si se quiere dar soporte a cualquier atributo de calidad. Sin embargo, su utilización durante la evaluación de una forma efectiva es posible, si se combina con otras estrategias (y esto es lo que proporciona el método CaLiPro).

4.2.1. Requisitos del método de evaluación

En resumen, los requisitos que debe cumplir el método respecto a la evaluación son los siguientes:

- Evaluar atributos de calidad de dominio (atributos de calidad operacionales o de ejecución) teniendo en cuenta la variabilidad que pueden tener en una línea de productos software.
- Método generalista que sirva para cualquier atributo de calidad operacional.
- Uso de métodos de evaluación y medición de atributos de calidad existentes.
- Reducir el esfuerzo de evaluación frente a evaluar cada producto.
- Unido a la evaluación y a los resultados que se obtienen de la misma, surge la oportunidad de facilitar la derivación para lo cual el método

también debe dar soporte.

4.2.2. Base del método de evaluación

La premisa de *Clements et al* que se cita a continuación sobre la evaluación de arquitecturas en las líneas de productos software es el punto de partida del método CaLiPro: si se identifica la variabilidad en la arquitectura software o en la implementación que afecta a los atributos de calidad, se puede concentrar la evaluación en esos puntos.

> "La evaluación de la arquitectura de producto es una variación de la evaluación de la arquitectura de línea de productos como la arquitectura de productos es una variación de la arquitectura de línea de productos y la extensión en la que la evaluación de producto es una evaluación separada y dedicada depende de la extensión en la que la arquitectura de producto se diferencia en formas que afectan a los atributos de calidad de la arquitectura de línea de productos" [Clements 01].

Para poder identificar la variabilidad que afecta a la calidad, hay que especificarla durante el desarrollo y relacionarla con la variabilidad que causa en la calidad. Luego hay que utilizar este modelo de variabilidad para reducir el esfuerzo de evaluación y evitar evaluar cada uno de los productos. Para ello, se proponen dos estrategias junto con el uso de métodos de evaluación o medición existentes. Las dos estrategias consisten en:

- Reutilización del modelo de evaluación: Crear un modelo de evaluación genérico que permita evaluar el diseño de varios o todos los productos.
- Reducción del alcance de la evaluación: Seleccionar un conjunto de productos de forma estratégica; el mínimo número de productos que permita extrapolar los resultados a la línea.

4.3. Modelado de características extendido con atributos de calidad

La base del método CaLiPro se encuentra en modelar la variabilidad durante la ingeniería de dominio para utilizar esta información a la hora de evaluar.

En concreto, se ha extendido el modelo de características, un modelo muy utilizado para modelar la variabilidad en las líneas de productos software. El objetivo de la extensión es permitir representar la variabilidad en los atributos de calidad. Para poder caracterizar los atributos de calidad, una adaptación del árbol de utilidad de atributos de calidad que se utiliza en las evaluaciones ATAM [Clements 02] ha sido también integrado en el modelo de características. El resultado de estas extensiones es el modelo CaFM (*CaLiPro Feture Model* o modelo de características CaLiPro).

En esta sección se detallan los requisitos que debía cumplir el modelo y se explican los modelos que se han extendido y se razona sobre la elección de los mismos.

4.3.1. Requisitos del modelo

Los requisitos que debe cumplir el modelado son los siguientes:

- Modelar la variabilidad funcional, arquitectural y de implementación, es decir, un modelo de variabilidad que se utilice en diferentes niveles de abstracción
- Modelar la variabilidad en los atributos de calidad
  - Modelar la variación indirecta, es decir, los impactos de la variabilidad funcional en la variabilidad de los atributos de calidad.
  - Permitir una caracterización de los atributos de calidad.
- Partir de modelos y enfoques existentes.

4.3.2. Modelo de características

Como base para su extensión se ha seleccionado el modelado de características puesto que es uno de los modelos más utilizados y casi un estándar de facto para el modelado de la variabilidad en las líneas de productos software. El modelado de características es considerado un requisito previo para la ingeniería de línea de productos software [Lee 02], porque juega un rol central, no sólo en el desarrollo de activos reutilizables sino también en la gestión y configuración de múltiples productos en un dominio. Además, el análisis de dominio orientado por características se ha utilizado mucho comparado con otras técnicas de análisis de dominio [Lee 02]:

1. Las características son un medio de comunicación efectivo entre participantes. Las características son abstracciones esenciales que ambos clientes y desarrolladores entienden y deben ser objetos de primera clase en el desarrollo software.
2. Las características son una forma efectiva de identificar la variabilidad (y la parte común) entre diferentes productos en un dominio. Las características son una forma natural e intuitiva de expresar la variabilidad.
3. El modelo de características puede proporcionar una base para desarrollar, parametrizar y configurar varios activos reutilizables (modelos de requisitos de dominio, modelo de arquitectura de referencia y componentes de código reutilizables).

Una característica es "cualquier concepto prominente y distintivo que es visible para varios *stakeholders*" [Lee 02]. Una característica también se define como "una propiedad de un sistema que es relevante para varios *stakeholders* y que es utilizado para capturar la parte común y discriminar entre sistemas" [Czarnecki 04]. Estos conceptos o propiedades también incluyen aspectos de calidad.

El modelado de características fue propuesto como parte del método FODA (Feature Oriented Domain Analysis) [Kang 90]. El modelo propuesto por FODA consiste en un diagrama de características, una jerarquía gráfica de *And/Or* de características. Las características pueden ser obligatorias, opcionales o alternativas. Las características opcionales se designan gráficamente con un pequeño círculo encima del nombre de la característica, como *Air Conditioning* (en la figura B.1). Las características alternativas se muestran como hijas de la misma característica padre, con un arco dibujado a lo largo de todas las opciones, como en el caso de *Transmission*. El arco significa que solamente una de las características debe ser elegida (equivale a la operación *xor* lógica). El

resto de características sin notación especial son todas obligatorias. Las reglas de composición se utilizan para definir la semántica existente entre características y que no se expresan en el diagrama: las relaciones de dependencia mutua (*Requires*) y exclusión mutua (*Mutex-with*). La información acerca del registro de *trade-offs*, razonamientos, justificaciones, etc. es también parte del modelo de características.

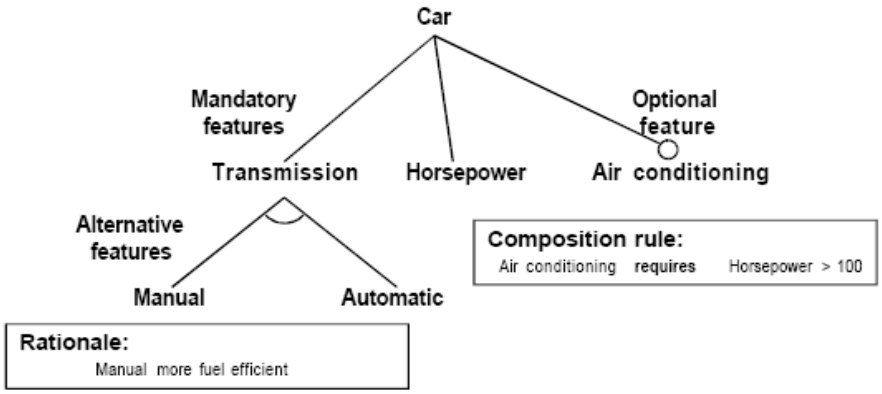


Figura 4.2: Ejemplo del enfoque FODA

A partir del método FODA, muchas extensiones y nuevos enfoques basados en el modelado de características han sido propuestos. En el anexo B se realiza un análisis de varios de estos enfoques y de los nuevos aspectos que se han ido añadiendo al modelo original.

Además de ser un concepto muy utilizado para modelar la variabilidad, las características también tienen otras vertientes que merece destacar. Por un lado, las características son utilizadas en dos dominios de utilización: dominio del problema y dominio de la solución. Y por otro lado, se está investigando sobre las interacciones de características, algo a considerar a la hora de definir los impactos. Este último punto también ha sido considerado para elegir el modelo de características como base del modelo CaFM.

4.3.2.1. Características y dominios de utilización

Una característica es un aspecto clave en las líneas de productos, tal como aparece en la definición *"...que comparten un conjunto común de características..."* [Clements 02].

Existen varias definiciones de característica, como por ejemplo:

- Una característica o *feature* es "un aspecto, calidad o característica prominente y visible para el usuario de un sistema o sistemas software" [Kang 90].
- "Una características es un incremento en la funcionalidad de un programa" [Batory 05].

Estas dos definiciones son bastante diferentes porque definen las características en diferentes dominios: el dominio del problema (características de dominio) y el dominio de la solución (características de implementación).

Las "características de dominio" son una visión centrada en el usuario de la funcionalidad del sistema, en el dominio del problema. Una característica es un conjunto de requisitos individuales dentro de una especificación de requisitos para el sistema, mientras que una "característica de implementación" es un subconjunto de la implementación del sistema. Una característica es un subconjunto de módulos de código asociados a una funcionalidad particular [Turner 99].

En los enfoques que usan características de dominio (como por ejemplo FODA, FORM, FeatuRSEB, *Generative Programming*, etc.) las características se utilizan para modelar el dominio pero no tienen representación de primera clase a nivel de implementación. Los enfoques que usan características de implementación (enfoques *Feature Oriented Programming (FOP)* como [Prehofer 97] o *Algebraic Hierarchical Equations for Application Design* (AHEAD) [Batory 04]) están orientados al código, al dominio de la solución.

Diferentes alternativas de características de implementación pueden realizar las características de alto nivel del dominio del problema. Además, una característica de dominio del problema puede ser implementado por uno o varias características de implementación [Pulvermueller 02]. Sin embargo, parece haber una falta de conexión entre los métodos orientados a implementación y los métodos orientados al modelado del dominio. Ciertos enfoques intentan construir un puente entre estos dos dominios: *Feature Engineering* (FE) [Turner 99], Feature-Based Product Derivation (FBPD) [Jansen 04] y *Feature-Solution Graph* (FSG) [de Bruin 01] son ejemplos de estos intentos.

En nuestro caso, una forma de modelar la variabilidad durante todo el ciclo de vida es necesaria pero no tanto desde un punto de vista de código o implementación, por lo que nos hemos centrado en los enfoques de dominio. En el anexo B se realiza una comparativa de varios enfoques basados en características que tienen un enfoque de dominio.

4.3.2.2. Interacción de características

"Una interacción de características ocurre cuando una o más características modifican o influencian otras características" [Liu 05]. La investigación en el área de las interacciones de características en las líneas de productos es bastante nueva. La investigación de la interacción de características o *feature interaction* ha sido tradicionalmente aplicada en otros dominios como los sistemas de telecomunicaciones.

Hay varios trabajos sobre interacciones de características que se centran en las relaciones de dependencias en el contexto de una línea de productos como por ejemplo: [Ferber 02a] y [Metzger 05]. [Metzger 05] propone un método de detección de interacciones basado en modelos para derivar dependencias en las líneas de producto. *Lee et al* [Lee 04] también han estudiado el rol de las dependencias en el modelado de interacciones de características en tiempo de ejecución, introduciendo la noción de activación y modificación de dependencias en diagramas de características.

También hay trabajos como el de *Liu et al* [Liu 05] que se centran en las interacciones estructurales y estáticas: "cómo una característica influencia (o cambia) el código fuente de otra" a la hora de componer productos utilizando el paradigma de *Feature Oriented Programming*.

Interacción de características y la calidad

La relación entre interacción de características y atributos de calidad ha sido mencionado en diferentes trabajos: [Stafford 01] manifestó como muchos atributos de calidad de sistemas críticos son expresiones de la interacción de características. En el ámbito de los servicios web [Weiss 05] también se menciona esta relación.

La interacción de características y la calidad en líneas de productos es mencionada en [Chastek 01] donde se propone el análisis de las interacciones de características; "... todas las posibles combinaciones de características deben ser analizadas, lo que es prácticamente imposible en sistemas con muchas características. Por tanto, el análisis debe centrarse en los atributos de calidad que son importantes para la línea de productos. El análisis identifica casos de uso que pueden afectar adversamente a esos atributos de calidad, y luego verifica si esos casos de uso se pueden dar con un conjunto de características dado". Sin embargo, no se propone ningún método para realizar esta tarea.

4.3.3. Enfoque CBFM y FeatuRSEB

Tras realizar un estudio de los enfoques basados en características que tienen un enfoque de dominio se ha optado por extender el enfoque CBFM (*Cardinality based Feature Modelling*) [Czarnecki 04]. Es uno de los enfoques más completos respecto a nuevos aspectos añadidos al modelo FODA clásico. Además, dispone de un metamodelo y de una descripción de semántica formal . También proporciona una herramienta *open-source* [Antkiewicz 04].

En el enfoque CBFM, hay tres novedades a considerar respecto al modelo de características clásico (FODA):

- *Cardinalidades de las características:* Las características se anotan con cardinalidades. Esta cardinalidad indica cuántas copias de una característica solitaria pueden hacerse [Czarnecki 02]. Las características obligatoria y opcional son características especiales con la cardinalidad [1..1] y [0..1].
- *Cardinalidades de grupo:* Conjuntos de características agrupadas que son anotadas con cardinalidades (como propone [Riebisch 02]).
- *Atributos:* Este concepto fue introducido por [Czarnecki 02] para representar un valor de un dominio grande o infinito como integers o strings.

También se van a adoptar las categorías de características que propone FeatuRSEB: características funcionales, arquitecturales y de implementación; y su filosofía de utilizar el modelo de características como modelo central de variabilidad durante el desarrollo.

4.3.4. Árbol de utilidad de atributos de calidad

A pesar de que las definiciones de características incluyan los atributos de calidad: Una característica o *feature* es "un aspecto, calidad o característica prominente y visible para el usuario de un sistema o sistemas software" [Kang 90]. En la práctica, no es posible describir correctamente un aspecto de calidad utilizando solamente una característica. Por lo tanto, se requiere de una forma de caracterizar o describir los atributos de calidad a la hora de representarlos como características.

El mecanismo seleccionado para describir y explicar los atributos de calidad

es una extensión del árbol de utilidad de atributos de calidad, pues tiene una estructura similar al modelo de características. Dicho árbol (utilizado en ATAM [Clements 02]) está orientado a caracterizar atributos de calidad para evaluar arquitecturas software. Su propósito es obtener una definición de los requisitos de calidad del sistema de forma práctica [Clements 02], por lo tanto, no es un intento de definir una taxonomía de atributos de calidad rigurosa. El árbol se estructura en tres niveles partiendo de la raíz denominada *Utility*: en el primer nivel están los atributos de calidad tales como rendimiento o seguridad; en el segundo nivel se encuentran las elaboraciones o *concerns* que detallan los atributos de calidad: por ejemplo, Tasa de transferencia (*throughput*) alta y latencia corta de transición punta a punta. En el tercer nivel se encuentran los escenarios que refinan o elaboran aún más los *concerns* o elaboraciones. Un escenario es una declaración de una expectativa respecto a un aspecto de calidad, por ejemplo *Internode message transfer completes <1 second.*

4.4. Herramientas desarrolladas

Para dar soporte al método elaborado, se han desarrollado dos herramientas una que da soporte a todo el proceso y otra sólo para la derivación. Para ello se ha partido de herramientas existentes (desarrolladas por equipos de investigación y con código abierto) y se han extendido y adaptado para dar soporte al método.

fmp-CaLiPro: Herramienta de soporte para la validación y la derivación

Se ha adaptado el **Feature Modeling Plug-in (fmp)** [Antkiewicz 04], un *plug-in* de Eclipse desarrollado por la universidad de Waterloo que permite editar y configurar modelos de características. Este *plug-in* ha sido extendido y adaptado para utilizarlo en la edición y configuración de modelos CaFM que incluyen aspectos de calidad, así como para facilitar la selección de productos.

En concreto la herramienta ha sido extendida para soportar la inclusión de características de calidad e impactos (tanto cualitativos como cuantitativos).

A partir de esta información es posible aplicar el algoritmo de selección de productos. Para ello, en la herramienta se ha añadido una funcionalidad que permite seleccionar los atributos a validar para aplicar el algoritmo de selección de productos y obtener la especificación de los productos a evaluar.

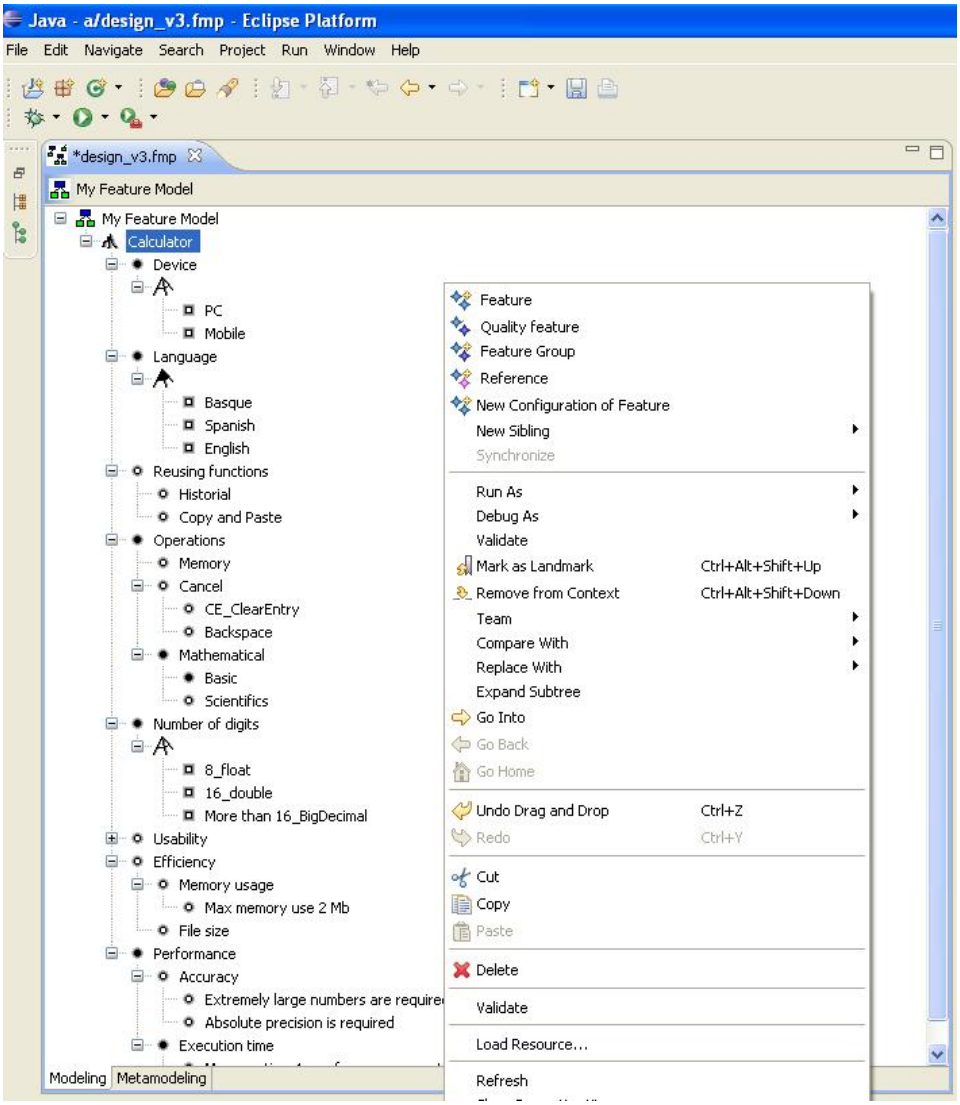


Figura 4.3: Imagen de pantalla de la herramienta fmp-CaLiPro

Una vez se han evaluado los productos seleccionados, la herramienta permite introducir los valores obtenidos por cada atributo de calidad, y se encarga de realizar las comparaciones de una forma automática, de detectar las interacciones y de proponer nuevos productos a evaluar si es necesario. Por último, cuantifica los impactos y permite al usuario introducir los niveles deseados para la derivación.

Respecto al soporte para la derivación, todavía no se ha implementado pero se va a partir de FAMA [Benavides 07], una herramienta desarrollada en la universidad de Sevilla y que tiene implementado el razonamiento automático mediante técnicas de inteligencia artificial tales como CSP. Se pretende extender esta herramienta e integrarla con la herramienta fmp-CaLiPro.

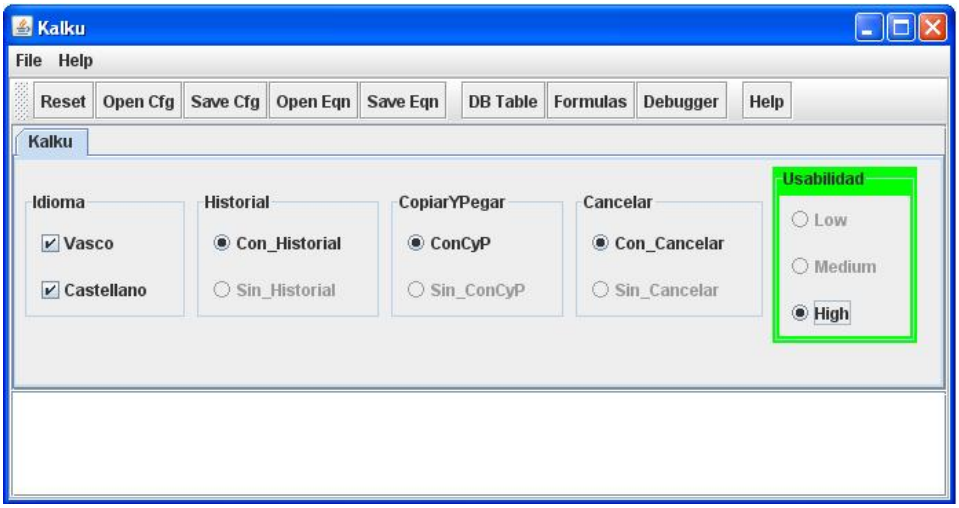


Figura 4.4: Imagen de pantalla de herramienta guiCa

Los desarrollos se han realizado en Java y se prevé ponerlos disponibles como herramientas *open source* cuando estén más maduros.

guiCa: Herramienta de soporte para la derivación

También se ha adaptado GuiDSL del AHEAD *tool suite* desarrollado por el grupo de investigación de investigación de *Product-line Architectures* de Don Batory de la Universidad de Texas. Dicha herramienta facilita la configuración o derivación de productos. La herramienta ha sido modificada para permitir mostrar aspectos de calidad y permitir una selección de características guiadas por atributos de calidad.

Para realizar un razonamiento automático durante la derivación es necesario disponer de técnicas de inteligencia artificial. Debido a los impactos introducidos los SAT Solvers (lógica formal) no sirven. Se requiere de otras técnicas tales

como CSP (*Constraint Satisfaction Problems*) como propone [Benavides 05]. En concreto se ha optado por utilizar la librería de java Choco que implementa técnicas de CSP y se está trabajando en ello.

4.5. Encaje con metodologías existentes

Independientemente del enfoque de modelo de características que se utilice, la extensión con atributos de calidad se puede implementar, siempre y cuando el modelo de características recoja variabilidad en aspectos de diseño e implementación.

En el caso de enfoques que utilizan un modelo de características de implementación tales como AHEAD y otros enfoques FOP, el modelo propuesto puede servir para llegar al modelo de características que se utiliza para codificar.

En el caso de enfoques basados en características de dominio tales como FODA, FORM, FOPLE, FeatuRSEB y CBFM se podrían extender los modelos existentes para añadir atributos de calidad e impactos. Y también puede ser necesario, ampliar el uso del modelo a otros momentos del desarrollo (para que el modelo también recoja características arquitecturales y de implementación).

Tampoco se impone un tipo de notación y/o modelos para modelar la variabilidad en el diseño, en la implementación o en los requisitos. Estos modelos serán complementarios al modelo de características, que será un modelo global de variabilidad.

4.6. Conclusiones

Este capítulo presenta el método CaLiPro y sus propiedades principales. El método está orientado a evaluar atributos de calidad de dominio en una línea de productos; para ello, propone un modelo de características extendido que permite capturar la variabilidad en los atributos de calidad junto a la variabilidad funcional. Partiendo de este modelo es posible centrarse en la variabilidad que afecta a la calidad y reducir el esfuerzo de evaluación.

El modelo para representar la variabilidad se basa en el modelo de caracte-

rísticas, uno de los modelos más utilizados para representar la variabilidad en las líneas de productos software. Durante este capítulo se explica en qué consiste el modelo de características, las diferentes definiciones de característica y las interacciones de las características. También se explican las razones de la selección de este modelo y de los enfoques concretos que se extienden.

En concreto se basa y extiende el metamodelo del enfoque CBFM y adopta la filosofía y categorías de FeatuRSEB. CBFM es un enfoque consolidado y ampliamente utilizado que además de proporcionar permitir describir cardinalidades de característica y de grupo o atributos dispone de una herramienta **Feature Modeling Plug-in (fmp)** [Antkiewicz 04]. Por otro lado, FeatuRSEB propone utilizar el modelo de características durante todo el desarrollo como modelo central que recoge la variabilidad en conjunto con el resto de artefactos en los que también se puede y se debe representar la variabilidad: requisitos, diseño, etc. El modelo también integra una extensión del árbol de utilidad de atributos de calidad que se utiliza en las evaluaciones ATAM para caracterizar los atributos de calidad. De este modo, es posible describir los atributos de calidad y su variabilidad. El resultado es el modelo CaFM que se detalla en el capítulo 5.

Por último, se describen las herramientas desarrolladas para dar soporte al método CaLiPro y se explica dentro de qué tipo de metodologías de desarrollo de línea de productos tiene encaje el método.

En el capítulo 6 se explica en detalle el proceso que hay que seguir para aplicar el método CaLiPro.

Capítulo 5

Modelo CaFM (Modelo de características extendido con atributos de Calidad)

5.1. Introducción

El presente capítulo presenta el modelo CaFM (Modelo de características CaLiPro). Este modelo se basa en el modelo de características y lo extiende para soportar atributos de calidad. En concreto, se ha partido del metamodelo de CBFM (*Cardinality Based Feature Model*) [Czarnecki 04] y de ciertos aspectos del enfoque FeatuRSEB, su filosofía de utilizar el modelo de características como modelo central durante todo el proceso y las tres categorías de características que propone: funcionales, arquitecturales y de implementación. El modelo de características se va aumentando (se agregan nuevas características) al ir bajando el nivel de abstracción (no sólo se usa en la fase de requisitos como en otros enfoques).

El otra vertiente del modelo, es la integración de una extensión del árbol de utilidad de atributos de calidad que se utiliza para caracterizar atributos de calidad con objeto de evaluarlos. Así como la definición de impactos para modelar la variación indirecta.

5.2. Modelo CaFM

El metamodelo del modelo de características extendido se muestra en la figura 5.1, con las extensiones realizadas al metamodelo de CBFM en otro color. El modelo extendido añade la categoría "características de calidad" para incluir elementos que faciliten la evaluación y derivación, lo cual facilita el razonamiento al integrar toda la información en el mismo modelo. El mecanismo seleccionado para describir y explicar los atributos de calidad es una extensión del árbol de utilidad de atributos de calidad, pues tiene una estructura similar al modelo de características. Dicho árbol (utilizado en ATAM [Clements 02]) está orientado a caracterizar atributos de calidad para evaluar arquitecturas software.

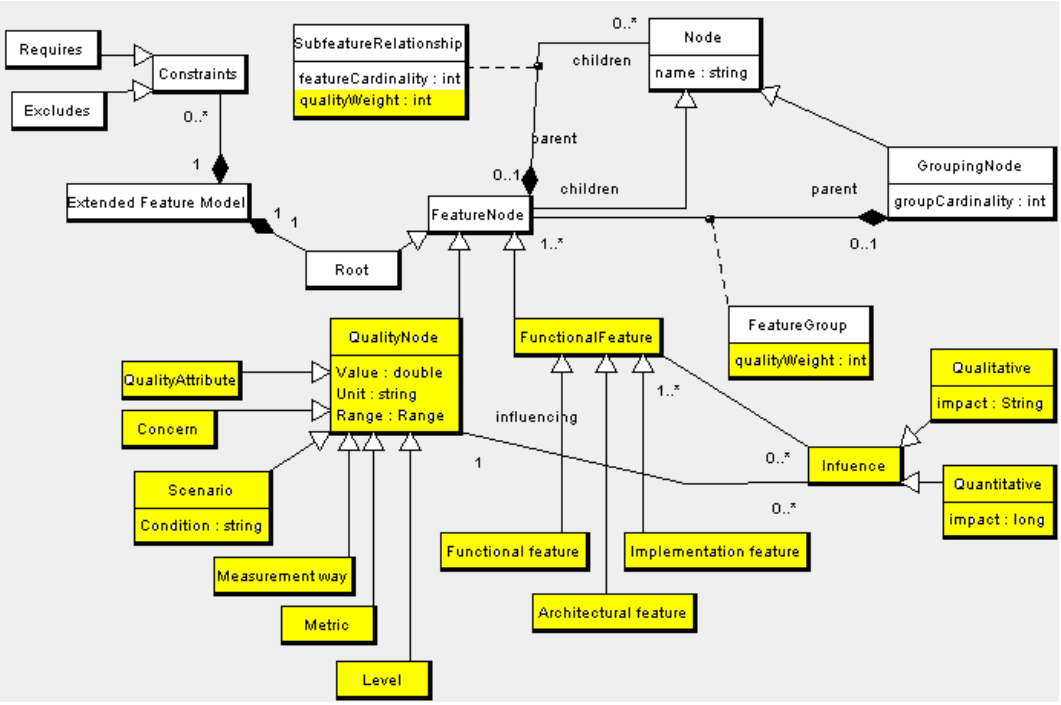


Figura 5.1: Metamodelo del modelo CaFM

Se han definido restricciones con OCL (*Object Constraint Language*) para especificar relaciones imposibles. Solamente debe haber una raíz y por debajo de está raíz puede haber características de calidad y funcionales. Dentro de una característica funcional no puede colgar una característica de calidad y viceversa. A partir de la raíz habrá dos ramas (una de características de calidad y la otra de características funcionales).

```
context Root inv c6: Root.allInstances->size=1
```

context Root inv c10: self.children->forAll (c|c. oclIsTypeOf (FunctionalFeature) or c.oclIsTypeOf(QualityNode) or (c.oclIsTypeOf (GroupingNode) and c. children ->forAll (oclIsTypeOf (FunctionalFeature))) or (c.oclIsTypeOf (GroupingNode) and c. children ->forAll (oclIsTypeOf (QualityNode))))

context FunctionalFeature inv c8: self .children ->forAll (c |c.oclIsTypeOf (FunctionalFeature) or (c.oclIsTypeOf(GroupingNode) and c.children ->forAll (oclIsTypeOf(FunctionalFeature))))

context QualityNode inv c8: self .children ->forAll (c |c.oclIsTypeOf (QualityNode) or (c.oclIsTypeOf(GroupingNode) and c.children ->forAll (oclIsTypeOf(QualityNode))))

Al extender el modelo de características, la estructura y filosofía del árbol se ha mantenido; sin embargo, la semántica cambia ya que el uso que se hace de las características de calidad difiere de las funcionales. En los enfoques basados en características, las categorías se utilizan principalmente para guiar la identificación de características [Kang 90], [Lee 02]. Sin embargo, la categoría de calidad se utiliza para facilitar otras tareas como la evaluación o la derivación. Para facilitar una diferenciación de las características de calidad, se propone el uso de un distintivo: diferente color, una etiqueta como por ejemplo una "q", una rama específica con atributos de calidad, etc.

Las dos principales extensiones que se han realizado en el modelo de características son, por una parte la inclusión de la categoría de características de calidad para caracterizar atributos de calidad con variabilidad; y por otro lado, la especificación de los impactos para representar la influencia de la variabilidad funcional en los atributos de calidad.

5.3. Características de calidad

Dentro de la categoría de calidad se definen tres **subcategorías** dependiendo de su uso. En la figura 5.2 se puede observar el uso de estas características:

- *Características de especificación:* Detallan atributos de calidad de la línea de productos. A este nivel los impactos son cualitativos. Este tipo de información es necesario para especificar toda la variabilidad de la línea.
- *Características de evaluación:* Se utilizan para caracterizar los atributos de calidad con objeto de evaluarlos. A este nivel los impactos siguen siendo cualitativos. Este tipo de información se utiliza para facilitar la evaluación.
- *Características de derivación:* Son el resultado de la evaluación partien-

do del modelo anterior (con características de evaluación) y facilitan la derivación. A este nivel los impactos son cuantitativos y sirven para guiar la derivación.

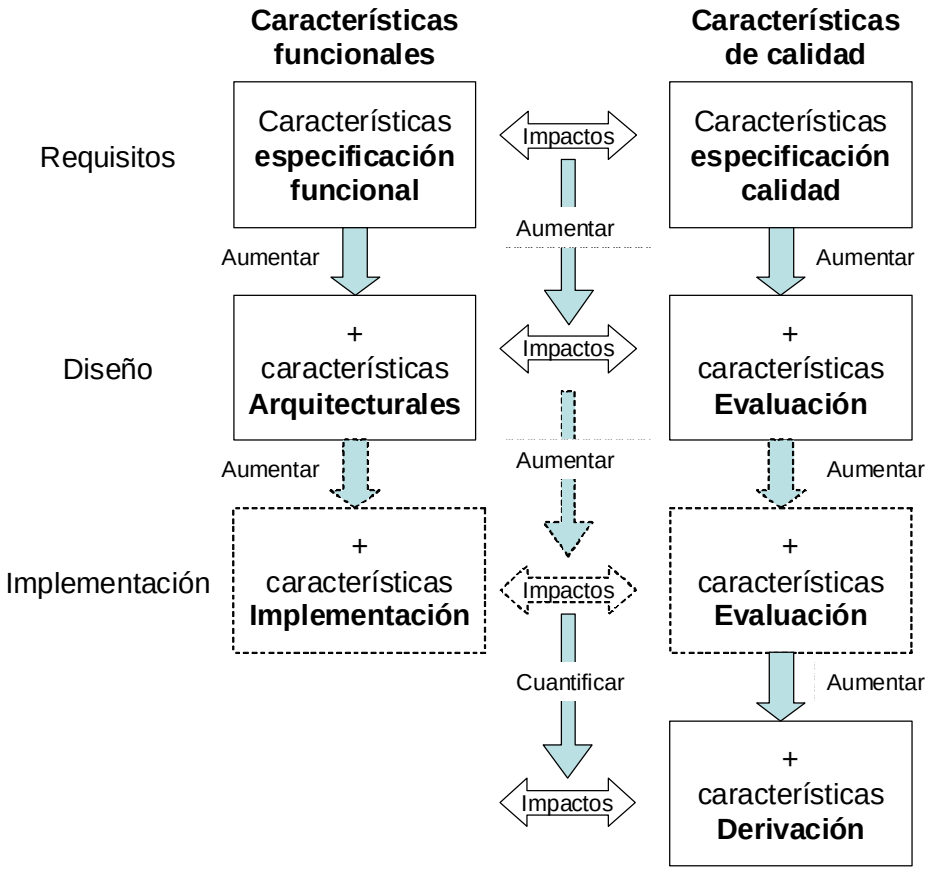


Figura 5.2: Categorías y subcategorías del modelo CaFM

Los atributos de calidad suelen ser requisitos bastante ambiguos y específicos de dominio, por lo que es conveniente detallarlos o caracterizarlos para poder evaluarlos. Las características de calidad van a permitir realizar esta caracterización; para ello, dentro de las características de calidad se pueden encontrar diferentes elementos o subtipos de característica (ver tabla 5.1). Estos subtipos de características también se observan en la extensión del metamodelo en la figura 5.1.

Tabla 5.1: Tipos de características de calidad

Tipo	**Subcategoría**	**Uso**	**Propiedades**	**Ejemplo**
Escenarios (restricciones en atributos medibles)	Evaluación	Se utilizan para representar restricciones o condiciones a cumplir.	Nombre: Expresión (puede estar expresado mediante estímulo, entorno y respuesta), sujeto de condición Operador Condición: <,>,=... Valor Unidad Rango o escala	Tamaño de fichero <100 Kb Sujeto de condición: Tamaño de fichero Operador: < Valor: 100 Unidad: Kb Rango o escala: -
Escenarios (subjetivos)	Evaluación	Se utilizan para representar aspectos a cumplir que no son directamente medibles. Es necesario cuantificar y definir una condición para estos escenarios en la fase de evaluación.	Nombre: Expresión, sujeto de condición Operador Condición: <,>,=... Valor Unidad Rango o escala	Soportar uso internacional (>80 en rango 0-100)
Concern o elaboración	Especificación y Evaluación	Se utilizan para representar aspectos de calidad detallando más. Pueden ser tanto de aspectos medibles como subjetivos.	Nombre Valor Unidad Rango o escala	Uso de memoria
Atributo de calidad	Especificación	Se utilizan para representar aspectos de calidad. Pueden ser tanto de aspectos medibles como subjetivos.	Nombre Valor Unidad Rango o escala	Usabilidad
Medidas	Evaluación	Las medidas están relacionadas directamente con la forma de medir los aspectos de calidad. Puede ser necesario introducir diferentes formas de medir. Son más de agrupamiento. Y se utilizan para obtener diferentes valores o clasificaciones para los atributos de calidad o elaboraciones.	Nombre Valor Unidad Rango o escala	Uso de memoria medido mediante una librería externa
Métricas	Evaluación	Las métricas son muy útiles durante el refinamiento para permitir la cuantificación de aspectos complejos o no medibles directamente.	Nombre Valor Unidad Rango o escala	Nº de idiomas
Niveles	Derivación	Se aplican sobre elaboraciones o atributos de calidad y consiste en dividir en grupos o niveles para facilitar la derivación. No pueden recibir impactos. Se utilizan los pesos para trasladar los impactos de los hijos hasta llegar al nivel en el que se definen los niveles.	Nombre Valor Unidad Rango o escala	Usabilidad alta, media y baja

Para describir estas características de calidad se utiliza el modelo *Quality feature tree*(árbol de características de calidad) que es una extensión del árbol de utilidad de atributos de calidad, al que se le han añadido nuevos subtipos y variabilidad (en el anexo D, se presenta información detallada de los cambios realizados al árbol de utilidad en nuestra propuesta). En la figura 5.3 se muestra un ejemplo del árbol de utilidad extendido. En concreto se ha añadido variabilidad, así como nuevos tipos de nodos.

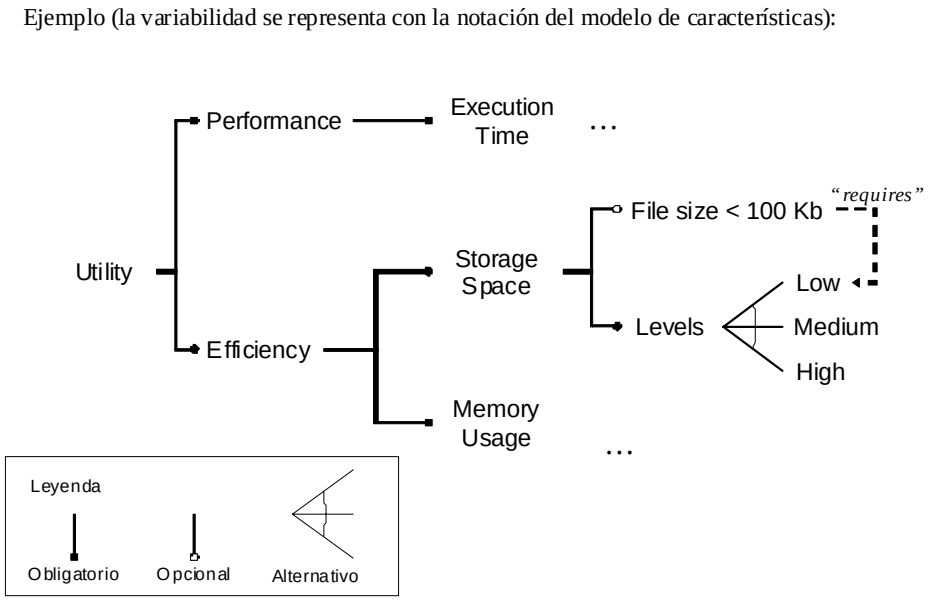


Figura 5.3: Ejemplo de un *quality feature tree*

5.4. Impactos

Una interacción de características ocurre cuando una o más características modifican o influyen a otras características [Liu 05]. A lo largo de la tesis, cuando se hace referencia a interacción de características se refiere a interacciones de características que causan variación en características de calidad (impactos). Los **impactos** son un caso particular de interacción de características que representan la variación indirecta (la variabilidad funcional que causa variación en los requisitos de calidad). Una vez que los requisitos tanto funcionales como de calidad están representados en el modelo de características extendido, se definen impactos de forma explícita entre las variantes funcionales, arquitectónicas y de implementación y los requisitos de calidad. Estos impactos pueden ser cualitativos o cuantitativos:

- Impactos cualitativos: Estos impactos se definen en un primer paso y son dependientes del diseñador y de los expertos. Los impactos pueden ser: ++,+,+-,-,- -. Siendo "- -": impacta muy negativamente, "-": impacta negativamente, "+": impacta positivamente, "+-": impacta pero no se sabe si positiva o negativamente y "++': impacta muy positivamente.
- Impactos cuantitativos: Son el resultado de evaluaciones de diseños de ciertos productos o del testeo o medición de estos productos.

El **impacto** denota cómo cambia la característica de calidad q a consecuencia de un cambio en la característica f_i. El cambio en la característica de calidad será un cambio en su valor (normalmente en su cantidad) mientras que el cambio en la característica f será cambiar de una variable a otra variante de la característica.

$$Impacto = \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})} \tag{5.1}$$

Siendo la característica f_i una característica normalizada con variantes alternativas $v_{ij} \in f_i$, $v_{ik} \in f_i$ y $j \neq k$. Por ejemplo: $f_1 = \{v_{11}, v_{12}, v_{13}\}$, el valor del atributo q para el producto 1 sería el siguiente, siendo q_{base} el valor de q del producto con el mínimo valor de q:

$$q_{prod_1} = q_{base} + \sum_{i=0}^{i=NumFeatures} \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})} \tag{5.2}$$

De este modo, se parte del valor mínimo (de un producto con ciertas variantes en las características) y se van añadiendo los impactos (el incremento o decremento causado por cambiar de variante en alguna de las características) hasta obtener el producto con las variantes deseadas.

El grado de interacción puede variar dependiendo del número de características involucradas. Así, un grado 1 es cuando una característica causa una variación en una característica de calidad, es decir, un impacto simple: $Interact(f_1; q_{f_1})$. Un impacto n se da cuando n características interaccionan con una característica de calidad; es un impacto de grupo, ya que el valor del impacto depende del valor de las n características involucradas y no es la suma de los impactos individuales: $Interact(f_1, f_2, f_3, ..., f_n; q_{f_1})$.

Si el cambio en el valor de la característica q cuando cambian dos características no es la suma de los impactos individuales (el cambio que causa una

característica más el cambio que causa la otra característica), hay una interacción grado 2 (o mayor): $Interact(f_i, f_l; q)$

$$si \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})\Delta f_l(v_{lm} \rightarrow v_{ln})} \neq \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})} + \frac{\Delta q}{\Delta f_l(v_{lm} \rightarrow v_{ln})} \quad (5.3)$$

Si hay interacciones de grado 2 o superior, hay que definir impactos grupales o impactos individuales pero condicionados (a los valores del resto de características con los que interaccionan, representados con el signo de la derivada parcial). Un impacto condicionado es el cambio en q a consecuencia de un cambio en f_i donde se cambia de la v_{ij} a v_{ik} manteniendo otras características con las que interacciona f_i constantes $(v_{lm}, v_{op}, ...)$.

$$Impacto = \frac{\partial q(v_{lm}, v_{op}, ...)}{\partial f_i(v_{ij} \rightarrow v_{ik})} = q_{f_i} = \partial_{f_i(v_{ij} \rightarrow vik)} q(v_{lm}, v_{op}, ...) \quad (5.4)$$

En el anexo F se explica con más detalle y ejemplos qué es una característica normalizada y las definiciones y el uso de los impactos.

5.5. Conclusiones

Como resumen, nuestro modelo de características extendido (CaFM) se basa en los siguientes aspectos:

Basado en	**Categorías de características**	**Nodos**	**Relaciones de descomposición**	**Otros aspectos**
CBFM y FeatuRSEB	Funcional Arquitectural Implementación Calidad	Cardinalidades de características	Cardinalidades de grupo	Una vista de calidad (características de calidad) Características especiales para propósitos de evaluación y derivación: niveles... Impactos y pesos

A pesar que los atributos de calidad aparecen en la definición de característica y se proveen categorías en las que tienen cabida, no se proporciona la necesaria caracterización de los atributos teniendo en cuenta su ambigüedad ni la forma de representar los impactos de las variantes funcionales en los atributos de calidad respondiendo a todos los tipos de variabilidad en los atributos de calidad mencionados previamente. Por ello, se ha extendido el modelo con el árbol de utilidad de atributos de calidad que permite caracterizar atributos de calidad.

El modelo CaFM permite representar la variabilidad tanto funcional como de calidad a lo largo del desarrollo y también permite especificar impactos de las características funcionales en las características de calidad. Utilizando este modelo se puede obtener información de qué variabilidad afecta a la calidad y centrarse en esa parte con objeto de reducir esfuerzos de evaluación. La especificación del modelo y su uso para la evaluación se explica en el siguiente capítulo.

Capítulo 6

Proceso CaLiPro

6.1. Introducción

En esta sección se detalla el proceso CaLiPro (ver figura C.2), explicando cada una de las fases del método propuesto con énfasis en aquellos puntos más interesantes. El proceso consta de tres fases: la especificación del modelo, el uso del modelo en la evaluación y en la derivación.

Ejemplo

Durante este capítulo se va a utilizar un ejemplo de un sistema no informático para ilustrar el método de manera sencilla y entendible. Se ha optado por el ejemplo del coche descrito en [Kang 90], introduciendo nuevos aspectos que permiten ilustrar la aplicación el método. Como se muestra en la figura 6.2, se han caracterizado los atributos de calidad (en color azul), como por ejemplo el respeto al medioambiente, el confort... utilizando el modelo CaFM (modelo de características extendido) que se propone en este trabajo.

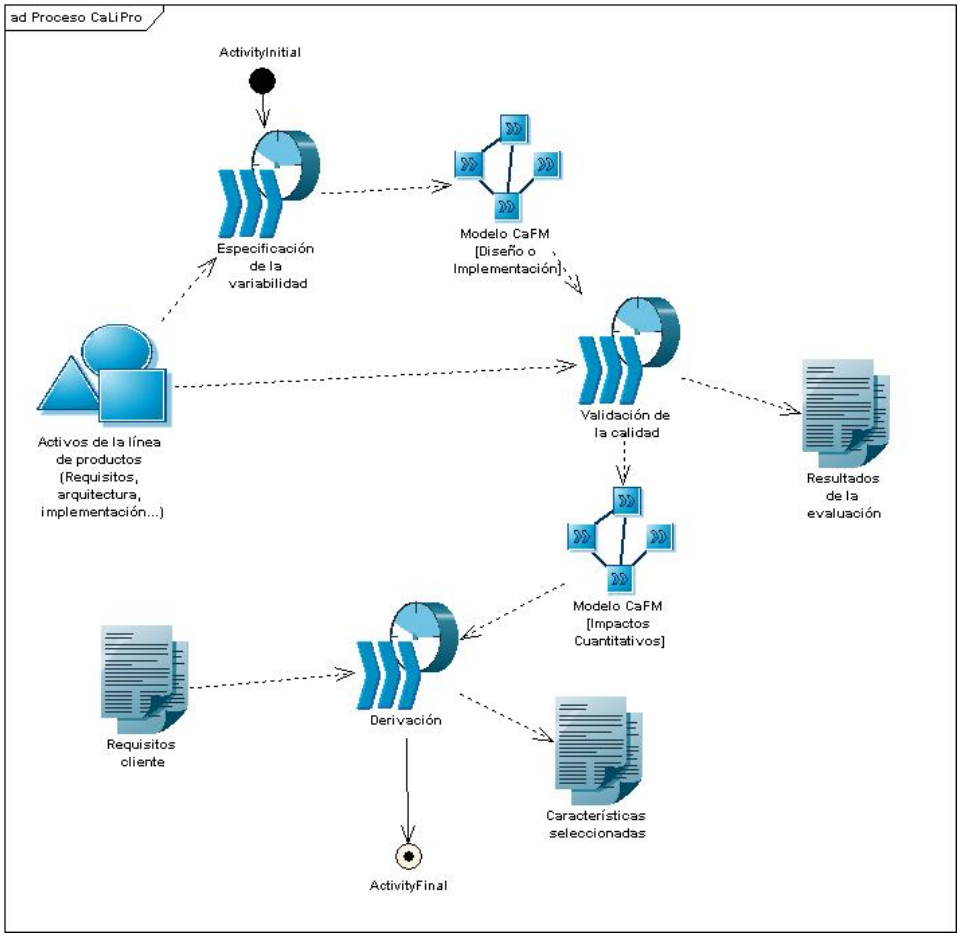


Figura 6.1: Proceso completo

6.2. Fase Especificación de la Variabilidad

El objetivo de esta fase es modelar las características de calidad, su variabilidad y los impactos para su posterior evaluación. El resultado de esta fase es un modelo de características extendido con información de variabilidad tanto funcional como de calidad en requisitos, diseño e implementación.

En la figura 4.1 se muestran gráficamente las diferentes fases del proceso de construcción. La secuencia del proceso depende del ciclo de vida que se siga a la hora de desarrollar la ingeniería de dominio. Las actividades principales se detallan a continuación:

- Definir la variabilidad funcional de la línea a nivel de requisitos, así como la variabilidad de calidad de la línea de manera concurrente.
- Aumentar el modelo de características extendido con características arquitecturales para representar la variabilidad en el diseño y caracterizar los aspectos de calidad de la línea. Se caracterizan los atributos de calidad para su evaluación mediante características de evaluación y se detallan los impactos. El modelo resultante puede ser la entrada para la evaluación de ciertos diseños de los productos.
- Aumentar el modelo de características extendido con características de implementación y caracterizar los aspectos de calidad. Se especifican las características de implementación y se definen los impactos respecto a las características de evaluación. Ésta también puede ser una entrada para la validación mediante testeo.

Ejemplo

En el coche los aspectos de calidad incluyen ser ecológicamente responsable o el confort. El modelo de variabilidad permite definir estos aspectos de calidad deseados y describirlos o caracterizarlos para el contexto de la línea que se está desarrollando (ver impactos y figura 6.2).

Impactos:

Air Conditioning impacts - on Low Fuel consumption

4WD impacts - on Low Fuel consumption

Manual impacts + on Low Fuel consumption

Automatic impacts - on Low Fuel consumption

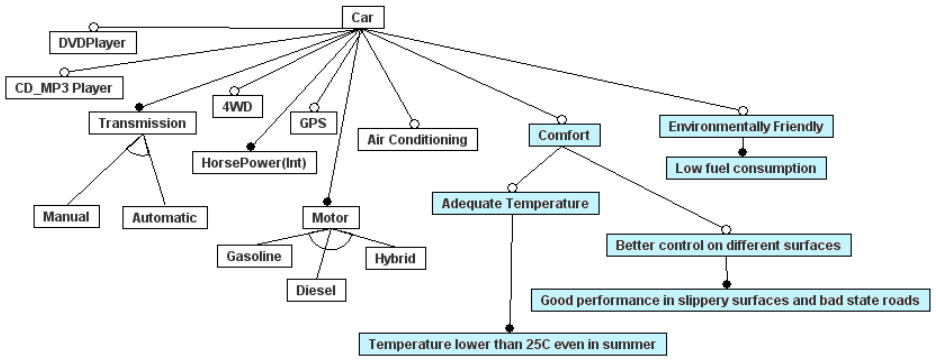


Figura 6.2: Modelo CaFM de la línea de productos de coches

Gasoline impacts - - on Low Fuel consumption

Diesel impacts - on Low Fuel consumption

Hybrid impacts + on Low Fuel consumption

HorsePower(int) ↑ impacts - on Low Fuel consumption

Air Conditioning impacts ++ on Adequate Temperature

4WD impacts ++ on Better control on different surfaces

Por ejemplo, seleccionar un coche con aire acondicionado impacta en el consumo de combustible de este coche porque el coche requiere más combustible cada vez que se pone en marcha el aire acondicionado. Por lo tanto, tiene una repercusión negativa a la hora de obtener un consumo bajo.

6.3. Fase Validación de calidad

Esta fase tiene como objetivo validar que la línea de productos permite obtener productos con los niveles de calidad requeridos. Para reducir los costes asociados a validar todos los productos de la línea se proponen dos estrategias:

- **Desarrollo de un modelo de evaluación genérico.** Cuando la línea está en su fase de diseño se puede realizar un modelo genérico de evaluación (dependiente del atributo a evaluar y del método de evaluación) que

representa a un conjunto de potenciales productos de la línea (pudiendo incluso valer para todos los productos).

- **Selección de productos a evaluar.** Es posible seleccionar un subconjunto de productos representativos que evaluados permitan extrapolar los resultados de calidad a todos los productos de la línea. Esta selección de productos se puede realizar tanto en fase de diseño (instanciando el modelo de evaluación) como cuando la línea está implementada (generación de productos y medición de la calidad en ejecución).

Las actividades de esta fase son las siguientes:

1. *Seleccionar los aspectos de calidad a validar:* Subconjunto de atributos que aparecen en el modelo de variabilidad que se quieren validar. Una vez seleccionados los atributos objetivo de la validación, se debe seleccionar el método concreto que se va a utilizar. Para ello se ha realizado una recopilación y comparativa de métodos de evaluación de arquitecturas que facilita la selección de un método apropiado a los atributo(s) a validar y el tipo de validación deseada (ver anexo E).
2. *Construir el modelo de evaluación genérico (solamente si se evalúa a nivel de diseño):* En caso de que la validación se realice en la fase de diseño de la línea mediante evaluación de arquitecturas software, hay que construir un modelo de evaluación o estimación genérico con variabilidad que pueda ser utilizado en un subconjunto de productos (pueden ser 1 o varios modelos genéricos dependiendo del grado y complejidad de la variabilidad de la arquitectura de la línea). El modelo a construir depende del método de evaluación que se vaya a aplicar.
3. *Seleccionar los productos que se van a testear o cuyos diseños se van a evaluar:* En esta fase se identifican productos representativos que una vez evaluados permitan extrapolar los resultados de la evaluación a todos los productos de la línea. Para ello se ha desarrollado un algoritmo que permite minimizar el número de productos elegidos. La selección de productos se puede hacer tanto en fase de diseño como una vez que la línea esté implementada. En casos excepcionales en los que la instanciación del modelo genérico o la generación del producto tengan coste muy reducido es posible que no se necesite realizar esta selección.
4. *Validación (testeo o evaluación de arquitecturas)*: La **validación** puede realizarse a nivel de diseño instanciando el modelo de evaluación genérico y a nivel de código vía testeos, mediante ejecución de los productos tras derivarlos. Una vez realizadas las validaciones de los productos hay que:
   a) Comparar resultados, detectar interacciones y decidir si es necesario validar más productos. Si es necesario hay que volver al punto 3. En

líneas en las que existan interacciones entre las características cuando impactan en un atributo de calidad, suele ser necesario validar más productos, para poder cuantificar los impactos o determinar con más exactitud qué características interaccionan.

b) Analizar todos los resultados: chequear que todos los escenarios obligatorios se cumplen en todos los casos y que los escenarios alternativos u opcionales también se cumplen según la especificación de los productos. Analizando los resultados debe ser posible asegurar que el rango de valores de los atributos de calidad de los productos de la línea se pueda obtener.

A continuación se detallan las dos estrategias para reducir el coste de evaluación que se proponen: construcción del modelo de evaluación genérico y la selección de los productos a testear.

6.3.1. Modelo de evaluación genérico

El modelo de evaluación genérico se realiza cuando la validación se hace sobre la arquitectura software de la línea. Los métodos de evaluación de arquitecturas software proponen el desarrollo de modelos que permiten de manera temprana evaluar si un producto va a cumplir los requisitos de calidad requeridos. Los modelos de evaluación a construir dependen del método y el atributo de calidad a evaluar. Por ejemplo, pueden ser gráficos de ejecución para evaluar el rendimiento siguiendo el método PASA (*Performance Analysis of Software Architectures*) [Williams 02], o pueden ser CDGs (*Component Dependency Graphs*) con probabilidades para evaluar la fiabilidad utilizando el método SBRA (*Scenario based Reliability Analysis*) para software basado en componentes [Yacoub 99] o modelos UML anotados con probabilidades para análisis de fiabilidad [Cortellessa 02], etc.

En el caso de las líneas de productos, la arquitectura software representa a todo el conjunto de productos que se pueden generar a partir de la misma. De la misma forma, el modelo de evaluación debe contemplar el conjunto de productos que la arquitectura está representando. Por lo tanto, es necesario incluir el concepto de variabilidad en el modelo de evaluación. Si el modelo de evaluación se construye utilizando lenguajes de modelado que dispongan de estándares o extensiones conocidos para la variabilidad (como por ejemplo UML [Gomaa 04a], [Clauß 01]), se pueden utilizar estas extensiones como guía para representar la variabilidad del modelo de evaluación. En otros casos puede

ser necesaria una notación *ad-hoc* para representar la variabilidad.

Para detectar y especificar la variabilidad en los modelos genéricos, los impactos y la documentación de la arquitectura software son vitales. Los impactos guían la identificación de la variabilidad que debe estar presente en el modelo.

La variabilidad del modelo debe ser trazable desde las características del producto. Para facilitar la trazabilidad, se construirá una tabla con las características y variantes que afectan al requisito de calidad y los aspectos variables en el modelo o modelos.

Ejemplo

En el siguiente ejemplo se muestra un modelo de evaluación genérico; en concreto es un gráfico de ejecución de un escenario crítico para la evaluación del rendimiento utilizando PASA. El escenario es el del refresco y para que el juego le parezca continuo al usuario, es necesario realizar 10 refrescos en un segundo. Durante el refresco, hay que calcular la nueva localización de cada uno de los elementos móviles, chequear y manejar las colisiones entre elementos y pintar todos los elementos. El modelo de rendimiento genérico ha sido especificado utilizando un gráfico de ejecución [Smith 02] (ver figura 6.3). Es parte del caso de estudio de juegos arcada que se detalla en el siguiente capítulo.

En el gráfico de ejecución, cada uno de los pasos de procesamiento es especificado como un nodo con nodos de repetición para expresar cuántas veces se repite el nodo. Cada paso tiene una tabla con sus requisitos de recursos. En este caso, la variabilidad es la siguiente:

- Tiempos de repetición de los nodos de repetición: Dependientes del número de objetos o *sprites* móviles (n_m), los *sprites* estacionarios (n_s) y el número de colisiones (n_c) que son dependientes del tipo de juego.
- Consumo CPU del paso de procesamiento "*paint*" es variante mayor o menor dependiendo de la resolución de las imágenes ($VarP$ almacena el consumo de CPU de la tarea "*paint*" especificado en las unidades de trabajo).
- Requisitos de recursos software del paso de procesamiento "*handling collision*" son variantes debido a que el soporte para sonido es opcional ($VarS_1$ almacena el consumo CPU de la tarea que es variante dependiente de si hay o no *feedback* de sonido y $VarS_2$ almacena el número de veces que se reproduce un sonido en el altavoz).

La sobrecarga de procesamiento (ver tabla 6.1) es una tabla de los requisitos

de recursos del ordenador por cada demanda de recursos software. Los recursos especificados en el ejemplo son *workUnit* (el número estimado de instrucciones de CPU para ejecutar una tarea simple), *get/free* (el número de llamadas a operaciones de gestión de memoria), *screen* (el número de veces que se "dibuja" en la pantalla) y *sound* (el número de veces que se reproduce un sonido en los altavoces). El tiempo de servicio para ejecutar 1 KInstrucciones ($VarS_1$), para dibujar en la pantalla ($VarS_2$) y para reproducir un sonido en los altavoces ($VarS_3$) también es variante dependiendo del dispositivo (un móvil o un pc).

Los aspectos variables mencionados se relacionan con las características en la matriz de trazabilidad (ver tabla 6.2) que relaciona las características y las variantes del modelo de evaluación.

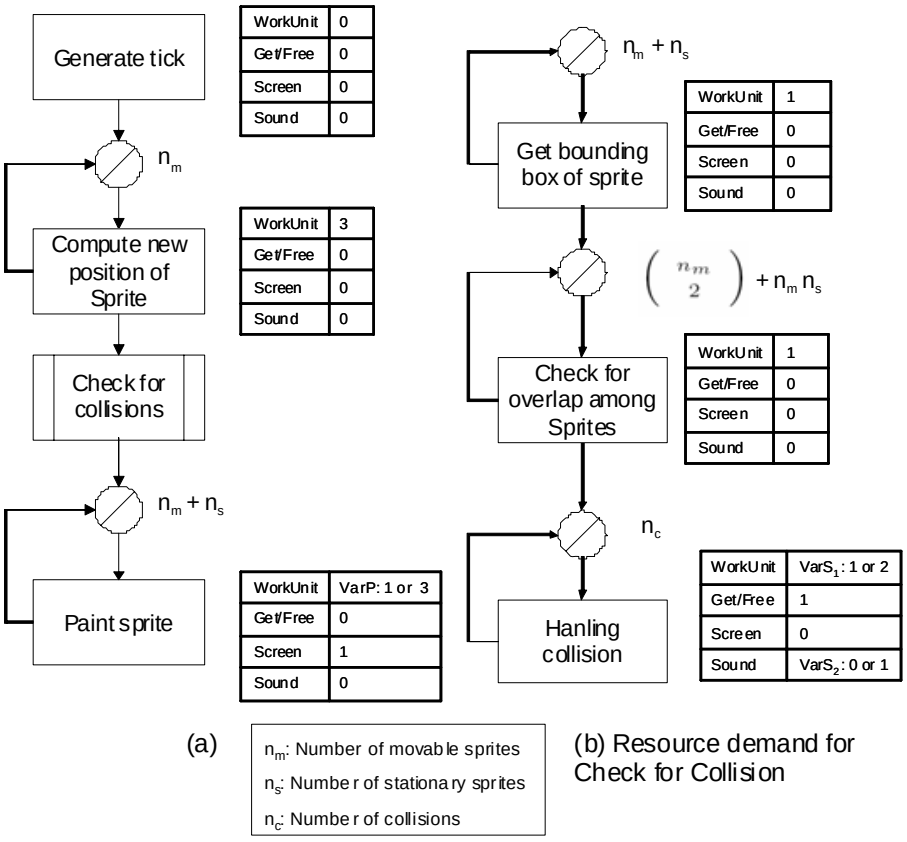


Figura 6.3: Modelo genérico: El gráfico de ejecución para el escenario de refresco

Tabla 6.1: Sobrecarga de procesamiento

Device	CPU	Display	Speaker
Quantity	1	1	1
Service Unit	Kinstr.	Units	Units
WorkUnit	20	0	0
Get/Free	2	0	0
Screen	1	1	0
Sound	1	0	1
Service Time	VarD1	VarD2	VarD3

Tabla 6.2: Matriz de trazabilidad de la variabilidad

Feature	Variante	nc	ns	nm	VarP	VarS1	VarS2	VarD1	VarD2	VarD3
Game type	Pong	1	4	3						
	Bowling	12	16	2						
	Blickles	3	36	4						
	SpaceWar	32	4	32						
Graphics Resolution	High				3					
	Low				1					
Sound	Yes					2	1			
	No					1	0			
Device	CPU							0,00001	0,002	0,002
	Mobile							0,00512	0,1	0,1

6.3.2. Seleccionar un subconjunto de productos a evaluar

Para seleccionar los productos se ha desarrollado un algoritmo. El algoritmo selecciona un subconjunto de productos que evaluados permiten obtener todo el rango de calidades de los productos de la línea. El algoritmo toma como entrada las características de la línea, la precisión que se desea (interacción simple o grupal), los impactos entre las características y los atributos de calidad a comprobar y selecciona los productos a evaluar siendo el número de productos seleccionado el mínimo posible para permitir detectar las interacciones existentes.

La selección de un grado u otro depende de la línea de productos y de las interacciones esperadas. Es aconsejable realizar una evaluación de productos incremental de modo que se comienza con los productos para detectar interacciones grado 2, y si tras analizar los resultados se detectan muchas interacciones, se decide evaluar también los productos necesarios para detectar interacciones grado 3 y así, hasta detectar todas las interacciones.

El algoritmo de selección de productos se basa en seleccionar productos con variantes diferentes, de modo que se puedan evaluar dichos productos y com-

parar los resultados. Para poder aplicar el algoritmo, primeramente hay que preparar el modelo y transformarlo. Los pasos son los siguientes:

1. Transformar o normalizar el modelo de características en una lista plana de conjuntos de opciones o variantes que afectan a los aspectos de calidad seleccionados previamente.
2. Eliminar las dependencias entre las variantes de la lista, creando diferentes listas.
3. Aplicar el algoritmo de selección de productos a las listas.
   a) Obtener los productos a testear.

Se ilustra el algoritmo con el ejemplo del coche.

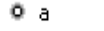
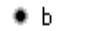
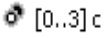
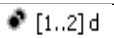
1. Transformación o normalización

El objetivo de las reglas de normalización es transformar los diferentes tipos de características en conjuntos de variantes alternativos y eliminar la estructura de árbol obteniendo una lista plana de conjuntos de variantes alternativos que afectan a uno o varios aspectos de calidad.

En la tabla 6.3 se explican las reglas de normalización para transformar los diferentes tipos de características en conjuntos de variantes alternativos. Los conjuntos se representan de la siguiente forma: $f_1 = \{V_{1,1}, V_{1,2}, V_{1,3}\}$, siendo *f1* una característica ya normalizada con tres variantes $(V_{1,1}, V_{1,2}, V_{1,3})$ de los que solamente se puede seleccionar uno. Algunas de las reglas son aplicables automáticamente pero en otras, llamadas manuales, es una persona la que tiene que valorar si se puede aplicar la regla y cómo aplicarla.

Los tipos de características considerados son aquellos se obtienen considerando las posibles cardinalidades de las características solitarias, cardinalidades de grupo y atributos de las características que pueden aparecer en el modelo CaFM.

Tabla 6.3: Reglas de normalización

Tipo de característica	**Cardinalidad**	**Ejemplo**	**Regla a aplicar**	**Resultado**	**Número de variantes**
Opcional (solitaria)	0..1		Crear un conjunto de dos variantes (sí y no).	a= a, no_a	$\sum_{c=0}^{1} \binom{1}{c} = \binom{1}{0} + \binom{1}{1} = 2$
Obligatoria (solitaria)	1..1		Excluir esta característica.	-	-
Opcional con posibilidad de clonar (solitaria)	0..n		Crear un conjunto con una variante para representar la posibilidad de no seleccionar y n variantes para cada copia.	c = no_c, c1, c2, c3	n (número de copias) +1
Obligatoria con posibilidad de clonar (solitaria)	1..n		Crear un conjunto con una variante por cada copia.	d = d1, d2	n = número de copias
Alternativa or xor (grupo)	1..1		Crear un conjunto con una variante por cada característica hija del grupo.	Grupo_e = f, g, h	$\sum_{c=1}^{1} \binom{nv}{c} = \binom{nv}{1} = nv$ Siendo nv: Número de características hijas del grupo
Or (grupo)	1..n		Crear un conjunto con variantes con todas las opciones.	Grupo_i = j, k, l, jk, jl, kl, jkl	$\sum_{c=1}^{n} \binom{nv}{c} = \binom{nv}{1} + \binom{nv}{2} + \ldots + \binom{nv}{n}$
			REGLA MANUAL. Si el número de características hijas que se seleccionan es suficiente para definir los impactos, crear un conjunto de nv variantes con el número de características que se pueden elegir.	Grupo_i=uno,dos,tres	nv: Número de características hijas del grupo

Alternativa opcional (grupo)	0..1	Grupo_m n o	Crear un conjunto con una variante por cada característica hija del grupo más una variante por la opción de no seleccionar ninguna de ellas.	Grupo_m = no_Grupo_m, n, o	$\sum_{c=0}^{1}\binom{nv}{c} = \binom{nv}{0} + \binom{nv}{1} = nv+1$
Or opcional	0..n	Grupo_p <1-2> q r s	Crear un conjunto con variantes con todas las opciones más una variante por la opción de no seleccionar ninguna de ellas.	Grupo_p = no_Grupo_p, q, r, s, qr, qs, sr	$\sum_{c=0}^{n}\binom{nv}{c} = \binom{nv}{0} + \binom{nv}{1} + \ldots + \binom{nv}{n} = \sum_{c=1}^{n}\binom{nv}{c} + 1$
			REGLA MANUAL. Si el número de características hijas que se seleccionan es suficiente para definir los impactos, crear un conjunto de nv variantes con el número de características que se pueden elegir más una variante por la opción de no seleccionar ninguna de ellas.	Grupo_p = no_Grupo_p, uno, dos, tres	nv+1
Característica solitaria con atributo (solitaria)	0..1 o 1..1 o 0..n o 1..n	t (STRING)	En caso de obligatorio, crear un conjunto con una variante y que los impactos sean función del valor de la variante. En los demás casos se aplican las reglas correspondientes.	t = no_t, t	Depende de cardinalidad, para el ejemplo 2 (igual que para las características opcionales)

El orden de aplicación de las reglas es *bottom-up*, y solamente habrá que aplicar más de una regla si en la estructura hacia arriba hay otras características que también afecten a las características de calidad seleccionadas. Si es así, primero se aplica la regla en la característica hoja más profunda, el resultado de aplicar esa regla será parte del conjunto de variantes que se obtenga de aplicar la siguiente regla y así sucesivamente.

El pseudoalgoritmo para aplicar las reglas de normalización a cada una de las características es el siguiente:

```
variables
  f: característica;
  set: conjunto variantes de f;
inicio
  si CardinalidadMínima(f) = 0 entonces añadir a set (``no_'' + nombre(f));
  si Solitaria(f) entonces
    si CardinalidadMáxima(f) = 1 y TieneAtributo(f) añadir a set (nombre(f));
    desde i:=0 hasta CardinalidadMáxima (f) haga añadir a set (nombre(f) + ``_'' + i);
  sino
    si CardinalidadMáxima(f) = 1 entonces
      desde i:=0 hasta NúmeroHijos (f) haga añadir a set (nombreHijo(f,i));
    sino
      ConseguirCombinacionesHijos (f, CardinalidadMáxima(f)) y añadir a set
    finsi
  finsi
fin
```

Si la característica es opcional (opcional solitaria, opcional con posibilidad de clonar, opcional alternativa u opcional or), hay que añadir al conjunto de variantes de la característica la variante de no seleccionar dicha característica. Si la característica es solitaria y no de grupo, hay que comprobar que no sea obligatoria, ya que si es obligatoria se obvia, salvo que tenga un atributo que impacte en algún atributo de calidad. Si es opcional con posibilidad de clonar, hay que añadir tantas variantes como posibles copias según la cardinalidad máxima. Si es una característica de grupo (alternativa u *or*), en el caso de las alternativas hay que añadir cada uno de los hijos al conjunto y en el caso de las *or* las combinaciones posibles dependiendo de cuál es la cardinalidad máxima.

Ejemplo

Ilustrando con el ejemplo:

Impactos:

Air Conditioning impacts - on Low Fuel consumption

4WD impacts - on Low Fuel consumption

Manual impacts + on Low Fuel consumption

Automatic impacts - on Low Fuel consumption

Gasoline impacts - - on Low Fuel consumption

Diesel impacts - on Low Fuel consumption

Hybrid impacts + on Low Fuel consumption

HorsePower(int)↑ impacts - on Low Fuel consumption

Air Conditioning impacts ++ on Adequate Temperature

4WD impacts ++ on Better control on different surfaces

Partiendo de esos impactos y teniendo en cuenta que los aspectos de calidad seleccionados son el confort y ser ecológicamente responsable, la lista que se obtiene tras la normalización es la siguiente:

AirConditioning = {AirConditioning, noAirConditioning}

4WD = {4WD, no4WD}

Transmission= {Manual, Automatic}

Motor = {Gasolina, Diesel, Hybrid}

HorsePower = {LessThan100 ($<$100), Between100And300($\leq 100 y \geq 300$), MoreThan300 ($>$300) }

Solamente hay una regla manual que hay que decidir si aplicar o no, el de número de caballos, se ha decidido hacer tres grupos con valores representativos.

Al aplicar las reglas también hay que tener cuidado con las dependencias y sobre todo tenerlas en cuenta al aplicar las reglas manuales. En el ejemplo, hay que tener en cuenta la dependencia: *Air Conditioning requires HorsePower>100*. Este número impone un límite que debería estar presente en los grupos que se hagan para facilitar convertir las dependencias.

---

2. Eliminar dependencias entre las características

Para poder aplicar el algoritmo es necesario eliminar las dependencias (*requires* y *excludes*) para que la selección de variantes al aplicar el algoritmo no sea dependiente de las mismas. Para ello, se selecciona una dependencia (origen y destino) y se crean dos listas, una cumpliendo la dependencia (solamente variante origen y variantes destino permitidos) y otra con el resto de posi-

bilidades (resto de variantes en la característica origen y todos los variantes destino permitidos)

Ejemplo

En el ejemplo hay dos dependencias:

`Hybrid requires Automatic`

`Air Conditioning excludes ‘‘<100’’ (horsePower)`

Al eliminar la primera dependencia surgen dos listas (una cumpliendo la dependencia y la otra con el resto de opciones) y de estas dos listas surgen otras dos por cada una:

Con dependencias	Primera dependencia eliminada (Hybrid requires Automatic)	Segunda dependencia eliminada (Air Conditioning excludes "<100" (horsePower))
$AirConditioning = \{AirConditioning, noAirConditioning\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Manual, Automatic\}$ $Motor = \{Gasolina, Diesel, Hybrid\}$ $HorsePower = \{< 100, \leq 100 y \geq 300, > 300\}$	$AirConditioning = \{AirConditioning, noAirConditioning\}$ $4WD = \{4WD, no_4WD\}$ $Transmission = \{Automatic\}$ $Motor = \{Hybrid\}$ $HorsePower = \{< 100, \leq 100 y \geq 300, > 300\}$	$AirConditioning = \{AirConditioning\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Automatic\}$ $Motor = \{Hybrid\}$ $HorsePower = \{\leq 100 y \geq 300, > 300\}$
		$AirConditioning = \{noAirConditioning\}$ $4WD = \{4WD, no_4WD\}$ $Transmission = \{Automatic\}$ $Motor = \{Hybrid\}$ $HorsePower = \{< 100, \leq 100 y \geq 300, > 300\}$
	$AirConditioning = \{AirConditioning, noAirConditioning\}$ $4WD = \{4WD, no_4WD\}$ $Transmission = \{Manual, Automatic\}$ $Motor = \{Gasolina, Diesel\}$ $HorsePower = \{< 100, \leq 100 y \geq 300, > 300\}$	$AirConditioning = \{AirConditioning\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Manual, Automatic\}$ $Motor = \{Gasolina, Diesel\}$ $HorsePower = \{\leq 100 y \geq 300, > 300\}$
		$AirConditioning = \{noAirConditioning\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Manual, Automatic\}$ $Motor = \{Gasolina, Diesel\}$ $HorsePower = \{< 100, \leq 100 y \geq 300, > 300\}$

Una vez eliminadas las dependencias quedan cuatro listas.

3. Aplicar algoritmo

El algoritmo consiste en seleccionar aquellos productos con las combinaciones que permitan detectar las interacciones. Por lo tanto, es necesario seleccionar el grado de interacciones a detectar, ya que dependiendo del grado el número de productos será diferente. Si el grado de interacción es 1 (es decir las características que impactan en la calidad no interaccionan entre ellas), por cada *feature* o característica f_i, se genera un producto con cada variante V_{ij} de esta característica, mientras que las demás características tienen una configuración de variantes igual. Con grado n (n características interaccionan), hay que seleccionar los productos de modo que todas las combinaciones de n

características posibles sean consideradas.

Siendo g: grado, f: feature y nf(f): NumeroVariantes de f.

```
int NumOpciones (feature f, grado g)
{ if (g==1) return (nv(f))
  else return (nv(f)*NumOpciones(f+1,g-1));}
```

El número de productos a seleccionar viene dado por la siguiente fórmula:

$$NumProductos = \sum_{f=1}^{NumFeatures-(g-1)} (NumOpciones(f,g) - NumOpciones(f-1,g-1) \tag{6.1}$$

El algoritmo y la forma de aplicarlo se explica con más detalle en el anexo F.

Una vez que se han obtenido los productos, hay que evaluarlos o testearlos utilizando para ello técnicas existentes. Si existen escenarios definidos para los atributos de calidad, hay que chequear que se cumplan. Para ello, muchas veces es necesario mirar los productos que se encuentran en los límites.

Ejemplo

Para ilustrar se va a aplicar el algoritmo manualmente, aunque ya se ha implementado en Java y se puede hacer automáticamente.

Primeramente se ordena de más variantes a menos:

$HorsePower = \{\leq 100 y \geq 300, > 300\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Automatic\}$ $Motor = \{Hybrid\}$ $AirConditioning = \{AirConditioning\}$
$HorsePower = \{< 100, \leq 100 y \geq 300, > 300\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Automatic\}$ $Motor = \{Hybrid\}$ $AirConditioning = \{noAirConditioning\}$
$HorsePower = \{\leq 100 y \geq 300, > 300\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Manual, Automatic\}$ $Motor = \{Gasolina, Diesel\}$ $AirConditioning = \{AirConditioning\}$
$HorsePower = \{< 100, \leq 100 y \geq 300, > 300\}$ $4WD = \{4WD, no4WD\}$ $Transmission = \{Manual, Automatic\}$ $Motor = \{Gasolina, Diesel\}$ $AirConditioning = \{noAirConditioning\}$

En el ejemplo, para grado 2, el número de productos a evaluar es: 2*2-0 + 3*2-0 + (2*2-0+2*2-2+2*2-2) + (3*2-0+2*2-2+2*2-2) = 28 productos de un total de 400 (un 7 % de todos los productos).

Si el grado deseado es 2, por cada *feature* o característica f_i, se generan productos con cada variante V_{ij} de esta característica y combinaciones dos a dos de la variante con las variantes del resto de características (sólo las siguientes, las anteriores no). Se combinan las variantes en el mínimo número de productos. A partir de la segunda característica, habrá productos ya considerados.

Los productos obtenidos se pueden ver en la tabla 6.4.

Validación

Una vez que los productos han sido seleccionados, se derivan los diseños o los productos y se miden los aspectos de calidad (en este caso el consumo de combustible). Para ello, se utilizan los métodos tradicionales (para un solo sistema) existentes para evaluar arquitecturas o medir aspectos de calidad (seleccionados en pasos anteriores).

En este caso testear los productos consiste en obtener el consumo de combustible de cada coche. Esta tarea puede realizarse midiendo un coche de este tipo o también puede ser un valor estimado por expertos basándose en datos previos, en estudios, etc. El grado de confort y las elaboraciones en las que se divide son más difíciles de medir, por lo que el resultado puede ser un valor estimado por expertos u obtenido mediante encuestas.

Una vez que se obtengan los valores del consumo de combustible, se pueden comparar para ver si hay interacciones. Se pueden ver los consumos en la tabla 6.4.

Hay que restar el resultado de todos los productos con una variante diferente y comparar los resultados. Como por ejemplo para ver si la potencia (*HorsePower*) interacciona con 4WD, se pueden comparar los productos 2-1 (con 4WD) vs 4-3 (sin 4WD); en este caso las restas de los consumos son iguales: $Consumo(Prod2) - Consumo(Prod1) = 2500 - 2450 = Consumo(Prod4) - Consumo(Prod3) = 2300 - 2250$. Esto significa que 4WD y la potencia en caballos no interaccionan.

Detectar interacciones

Tras realizar todas las comparaciones, las interacciones detectadas son las siguientes (ver figura 6.4):

Tabla 6.4: Productos seleccionados y consumo de dichos productos

Num	Características	Consumo
1	Between100and300 4WD Automatic Hybrid AirConditioning	2450
2	MoreThan300 4WD Automatic Hybrid AirConditioning	2500
3	Between100and300 no4WD Automatic Hybrid AirConditioning	2250
4	MoreThan300 no4WD Automatic Hybrid AirConditioning	2300
5	LessThan100 4WD Automatic Hybrid noAirConditioning	2300
6	Between100and300 4WD Automatic Hybrid noAirConditioning	2350
7	MoreThan300 4WD Automatic Hybrid noAirConditioning	2400
8	LessThan100 no4WD Automatic Hybrid noAirConditioning	2100
9	Between100and300 no4WD Automatic Hybrid noAirConditioning	2150
10	MoreThan300 no4WD Automatic Hybrid noAirConditioning	2200
11	Between100and300 4WD Manual Gasoline AirConditioning	4130
12	MoreThan300 4WD Manual Gasoline AirConditioning	4330
13	Between100and300 no4WD Automatic Diesel AirConditioning	3375
14	MoreThan300 no4WD Automatic Diesel AirConditioning	3500
15	Between100and300 no4WD Manual Gasoline AirConditioning	4130
16	Between100and300 4WD Automatic Diesel AirConditioning	3675
17	Between100and300 4WD Automatic Gasoline AirConditioning	4200
18	Between100and300 4WD Manual Diesel AirConditioning	3626
19	LessThan100 4WD Manual Gasoline noAirConditioning	3780
20	Between100and300 4WD Manual Gasoline noAirConditioning	3880
21	MoreThan300 4WD Manual Gasoline noAirConditioning	4080
22	LessThan100 no4WD Automatic Diesel noAirConditioning	3100
23	Between100and300 no4WD Automatic Diesel noAirConditioning	3175
24	MoreThan300 no4WD Automatic Diesel noAirConditioning	3300
25	LessThan100 no4WD Manual Gasoline noAirConditioning	3430
26	LessThan100 4WD Automatic Diesel noAirConditioning	3400
27	LessThan100 4WD Manual Diesel noAirConditioning	3350
28	LessThan100 4WD Automatic Gasoline noAirConditioning	3850

- Air conditioning + Motor
- Motor + Transmisión
- Horsepower + Motor
- 4WD + motor

Por ejemplo, al comparar los productos 3-9 (con motor híbrido) versus los productos 13-23 (con motor diésel), se puede detectar una interacción entre el tipo de motor y el aire acondicionado. Los productos 3 y 9 sólo se diferencian en que uno tiene aire acondicionado y el otro no, los productos 13 y 14 también se diferencian en lo mismo. A su vez, los productos 3 y 13 (y 9 y 23) sólo se diferencian en el tipo de motor. Al comparar los consumos: $Consumo(Prod3) - Consumo(Prod9) = 2250 - 2150 = 100 \neq Consumo(Prod3) - Consumo(Prod9) = 3375 - 3175 = 200$ se observa que existe una interacción y que el impacto de introducir aire acondicionado es dependiente del tipo de motor. De esta forma, el impacto será de 100 cuando el motor sea híbrido y será de 200 en un motor diésel.

Si hay que realizar un análisis de los límites, es necesario cuantificar los impactos definidos previamente en el modelo de variabilidad. En este caso no

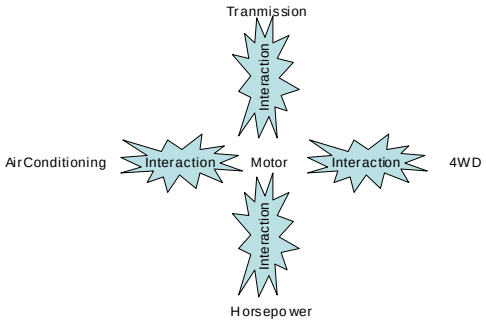


Figura 6.4: Interacciones detectadas en el ejemplo del coche

hay ningún escenario a chequear, pero vamos a suponer que existe el siguiente escenario "Consumo <4380".

En el ejemplo, a partir de las mediciones realizadas se puede cuantificar el impacto en el consumo de carburante. Los impactos no pueden ser individuales, ya que existen interacciones. Además, el motor está involucrado en todas las interacciones, lo que produce interacciones entre interacciones. Por esta razón, hay que aplicar impactos individuales pero condicionales.

Un impacto individual condicional determina el impacto de introducir una variante manteniendo el resto de variantes estables y según la condición. Los impactos son relacionales, el impacto que supone cambiar de variante. Así en el ejemplo los impactos son los siguientes:

$AirConditioning + (Hybrid) \rightarrow +100$

$AirConditioning + (Diesel) \rightarrow +200$

$AirConditioning + (Gasoline) \rightarrow +250$

$Automatic + (Diesel) \rightarrow +50$

$Automatic + (Gasoline) \rightarrow +70$

$Manual + (Diesel) \rightarrow -50$

$Manual + (Gasoline) \rightarrow -70$

$Between100and300 + (Hybrid) \rightarrow +50$

$Between100and300 + (Diesel) \rightarrow +75$

$Between100and300 + (Gasoline) \rightarrow +100$

$MoreThan300 + (Hybrid) \rightarrow +100$

$MoreThan300 + (Diesel) \rightarrow +200$

$MoreThan300 + (Gasoline) \rightarrow +300$

$4WD + (Hybrid) \rightarrow +200$

$4WD + (Diesel) \rightarrow +300$

$4WD + (Gasoline) \rightarrow +350$

$Diesel + (noAirConditioning + Automatic + LessThan100 + no4WD) \rightarrow +1000$

$Gasoline + (noAirConditioning + Automatic + LessThan100 + no4WD) \rightarrow +1400$

Partiendo del mínimo (Hybrid, LessThan100, noAirConditioning, no4WD y Automatic) se pueden ir aplicando los impactos uno a uno (cambiando cada vez sólo una variante) hasta obtener el producto deseado y su nivel de consumo. De este modo es posible obtener el producto con consumo máximo (MoreThan300 4WD Automatic Gasoline AirConditioning) que consume 4400. En este caso, este producto no ha sido evaluado pero se puede obtener su nivel de consumo a partir de los impactos. El escenario "Consumo <4380" no se cumple, así que si estuviéramos hablando de una línea de productos software podría ser necesario un rediseño.

6.4. Fase Derivación

Dentro de esta fase hay dos etapas diferenciadas: una radica en añadir información relativa a la calidad para facilitar la derivación y la otra consiste en utilizar esa información durante la derivación.

La primera etapa se basa en aumentar el modelo de características extendido con características para facilitar la derivación posterior: niveles o grupos para facilitar la derivación guiada por calidad e impactos cuantificados.

Los desarrolladores pueden utilizar los impactos cuantificados y obtenidos previamente a partir de los resultados de las evaluaciones y los testeos si se dispone de ellos o realizar una cuantificación mediante su experiencia y la ayuda de expertos. También deben definir los niveles o grupos en los aspectos de calidad. Los niveles se definen según la precisión deseada y teniendo en cuenta los impactos cuantitativos existentes. Si es necesario, se definen los pesos para

trasladar los impactos de los hijos hasta llegar a las características superiores en el que se definen los niveles.

Ejemplo

En el ejemplo, es posible definir niveles o grupos de niveles para el consumo de carburante: alto, medio, bajo. El producto con el mínimo consumo gasta 2100 y el que más consume 4400. Con estos datos se pueden hacer grupos con niveles:

- Nivel consumo bajo: [2100-2867)
- Nivel consumo medio: [2867-3633)
- Nivel consumo alto: [3633-4400]

La segunda etapa se realiza durante la derivación de los productos. Las características de calidad tienen ciertos aspectos que hay que considerar a la hora de derivar o configurar productos. Al elegir una característica funcional (no de calidad), ya sea funcional, arquitectónica o de implementación, se selecciona que dicha característica esté presente en el producto: en el caso de una característica funcional será una funcionalidad y en el caso de una característica arquitectural o de implementación una decisión o variante de diseño o implementación que se incluirá en la implementación del producto. Sin embargo, al seleccionar una característica de calidad, lo que se selecciona no es tan tangible. En un nivel superior, al seleccionar una característica -por ejemplo una característica atributo de calidad- lo que se expresa es que dicha característica es importante durante la derivación y se utilizará su caracterización con objeto de expresar con más detalle los requisitos para ese atributo. Por ejemplo, al seleccionar la característica opcional "*Performance*", se especifica que existen requisitos de "*Performance*" y que es algo a considerar. En su caracterización se dispondrá de más elementos para determinar esos requisitos, por ejemplo un escenario opcional que impone una restricción sobre el tiempo de ejecución. Todas las características de calidad de alto nivel (atributos de calidad, elaboraciones, medidas...) tienen que disponer de características que permitan representar los requisitos para esa característica, mediante los niveles y los escenarios. Los escenarios permiten representar restricciones o condiciones que hay que cumplir, y los niveles permiten representar grupos de valores dentro de la característica.

Los escenarios tienen una condición numérica que puede ser explícita o implícita para expresar una restricción. En el caso de que sea implícita, puede que

sea necesario cuantificar ese escenario y definir la condición de forma explícita. Los escenarios con condiciones implícitas suelen ser muy comunes cuando se detallan atributos de calidad que no se pueden medir directamente tales como la usabilidad. Por ejemplo, "*Several languages are supported*" puede ser un escenario con una condición implícita; para poder decir que este escenario se cumple ¿cuántos lenguajes son necesarios? Habría que definir una condición explícita: teniendo en cuenta que los lenguajes o idiomas incluidos tendrán un impacto en este escenario, se puede decir que este escenario se cumplirá cuando sea >80 de un rango de 0-100. De este modo, en una línea de productos que puede tener de 1 a 3 idiomas, 3 idiomas impactarán 100 y se cumplirá, 2 idiomas impactarán 50 y no se cumplirá (pero el valor 50 será considerado) y 1 idioma impactará 0 y no se cumplirá. Por defecto, la condición será que sea = 100 de un rango de 0-100, salvo que se especifique lo contrario.

En la tabla 6.5 se detallan diferentes opciones a la hora de utilizar el modelo durante la derivación.

Tabla 6.5: Diferentes escenarios durante la derivación

	Impactos	**Ejemplo**	**Al seleccionar funcionales. Comprobar**	**Diferentes opciones**
Escenario con condición explícita	Base impact X Historial (opt) impact B Grupo alternativo: Op1 impact Y Op2 impact Z Op3 impact A	Tamaño fichero <C Kb (opcional)	X + Z + B <C Si no se cumple, se deshabilita escenario	Primero se selecciona que se desea cumplir el escenario. a)Ir seleccionando e ir comprobando: dar error si no se cumple tras una selección b) Deshabilitar todas las opciones imposibles c) Obtener una lista de todas las opciones posibles Las opciones b y c pueden llegar a ser muy costosas computacionalmente
Niveles	Base impact X Historial (opt) impact B Grupo alternativo: Op1 impact Y Op2 impact Z Op3 impact A	Nivel 1 Nivel 2 Nivel 3	X + Z + B cuánto es y ver a que nivel corresponde. Seleccionar dicho nivel	Primero se selecciona un nivel deseado. a) Ir seleccionando e ir comprobando: dar error si se sobrepasa nivel (si se está por abajo dará error al salir pero de momento no) b) Deshabilitar todas las opciones imposibles c) Obtener una lista de todas las opciones posibles Las opciones b y c pueden llegar a ser muy costosas computacionalmente
Impactos cualitativos y pesos	Base impact a Opc (opt) impact b Grupo alternativo: Op1 impact c Op2 impact d Op3 impact e	Nivel 1 Nivel 2 Nivel 3	1) Cuantificar impactos 2) Obtener un valor en los *features* a los que llegan los impactos 3) Teniendo en cuenta los pesos, trasladar valor a nivel en el que están los grupos 4) Determinar a que grupo corresponde y seleccionar	Primero se selecciona un nivel deseado. a) Ir seleccionando e ir comprobando: dar error si se sobrepasa nivel (si estamos por abajo dará error al salir pero de momento no) b) Deshabilitar todas las opciones imposibles c) Obtener una lista de todas las opciones posibles Las opciones b y c pueden llegar a ser muy costosas computacionalmente

Durante la derivación, los escenarios serán seleccionados, mientras que los niveles serán una fuente de información: al ir seleccionando características funcionales se calculará el nivel y se marcará. Pero los niveles también podrán ser seleccionables, de modo que sea posible ver qué características funcionales pueden elegirse y cuáles no para obtener ese nivel de calidad.

A la hora de derivar, es necesario e imprescindible disponer de soporte, es decir, alguna herramienta. Si no puede resultar realmente difícil razonar sobre la calidad y toda la información disponible dejaría de tener utilidad.

En este sentido, existen trabajos como la herramienta *fmp* [Antkiewicz 04], la herramienta FAMA [Benavides 05] y la herramienta guidsl del AHEAD *tool*

suite [Batory 05] que ayudan durante la derivación. Se han tomado como base estas herramientas y se han extendido para dar soporte a la derivación teniendo en cuenta aspectos de calidad.

El razonamiento durante la derivación se simplifica al utilizar el modelo CaFM con elementos añadidos específicamente para que faciliten la derivación. Hay diversas situaciones que se quieren poder reflejar. En la 6.6 se detallan estas situaciones y la forma de representarlas y utilizarlas durante la derivación.

Tabla 6.6: Diferentes situaciones durante la derivación

Situación	**Ejemplo**	**Modelado**	**Uso en derivación**
Restricciones obligatorias	Uso de memoria <2Mb	Se modelan como escenarios obligatorios. E impactos cuantitativos.	Se chequean durante la evaluación, en derivación no importa ya que todos los productos tienen que cumplir dicha restricción.
Restricciones opcionales	Tamaño fichero <300 Kb	Se modelan como escenarios opcionales. Más impactos cuantitativos.	Durante la derivación, es necesario calcular la suma de los impactos cuantitativos y ver si se cumple.
Sin restricciones pero el nivel del atributo se considera importante	Rendimiento	Se modela utilizando niveles. E impactos cuantitativos.	Durante la derivación, se va calculando la suma de los impactos para ver en que nivel se encuentra.
Aspectos cualitativos	Usabilidad	Impactos cualitativos. Niveles. Pesos.	Previo a la derivación, se cuantificarán los impactos. Durante la derivación, se va calculando la suma de los impactos para ver en que nivel se encuentra.
Restricciones opcionales + niveles	Tamaño fichero <300 kb Se desean niveles	Niveles. Impactos cuantitativos. Escenario opcional. Dependencia entre escenario opcional y cierto nivel o niveles. CONDICIÓN: El valor de la restricción tienen que estar próximo a un límite.	Durante la derivación, se va calculando la suma de los impactos para ver en que nivel se encuentra y si cumple la restricción.

Ejemplo

En el ejemplo, se puede ir observando el nivel de consumo al ir seleccionando las características y también es posible seleccionar un nivel de consumo deseado.

Por ejemplo, al seleccionar diesel y AirConditioning (hay que sumar a la base mínima 2100, 1000 de diesel y 200 de AirConditioning) el nivel sería Medio (3300).

6.5. Conclusiones

En este capítulo se ha explicado el proceso para aplicar el método CaLiPro. El proceso consta de tres fase principales: la especificación de la variabilidad durante la ingeniería de dominio, la validación de la calidad utilizando el modelo con la variabilidad especificada y mediante estrategias que ayudan a reducir el esfuerzo de evaluación y la fase de derivación que aumenta el modelo a partir de los resultados de la evaluación para facilitar la derivación guiada por atributos de calidad.

Las dos estrategias utilizadas a la hora de reducir esfuerzos son la creación de un modelo genérico de evaluación que permite evaluar toda la línea y la selección de un subconjunto de productos para posteriormente extrapolar los resultados a la línea.

El método además de facilitar la validación de los aspectos de calidad, hace explícitos los impactos e interacciones existentes a la hora de lograr el nivel de calidad. Y esta información es muy útil no sólo para optimizar la evaluación sino para facilitar una derivación guiada por calidad.

En el siguiente capítulo, se resumen los casos de estudio en los que se ha aplicado en método CaLiPro.

Capítulo 7

Casos de estudio

7.1. Introducción

En este capítulo se presenta la aplicación del método en casos reales que han servido para extender y consolidar el método a partir de las lecciones aprendidas y validar los resultados obtenidos con el método.

El objetivo de la investigación era proporcionar un método que reduzca los costes de la validación de atributos de calidad y la derivación teniendo en cuenta aspectos de calidad. Para validar el método propuesto se han utilizado tres casos de estudio ya que son técnicas de investigación adecuadas cuando las preguntas de investigación "cómo" y "por qué" conciernen a un conjunto de eventos contemporáneos donde el investigador tiene poco o ningún control (en el sentido experimental) [Yin 94], como ocurre en este caso. La validación tiene dos vertientes:

1. ¿Se reduce el tiempo y esfuerzo necesario para validar la línea a través de este método?
2. ¿Aplicando el método propuesto se reduce el esfuerzo necesario para derivar productos con ciertos niveles de calidad?

Para poder aplicar y validar el método es necesario tener una línea de productos software con unas características concretas:

- Atributos de calidad variantes.
- Con el diseño y arquitectura software documentada o con código fuente

disponible o fácilmente codificable en un tiempo razonable. Para poder validar el método contra datos reales obtenidos mediante la evaluación de la arquitectura o la ejecución y el testeo de los productos.

Respecto a los atributos de calidad, además de tener variabilidad, es necesario tener diferentes tipos de atributos de calidad, tanto atributos medibles de forma objetiva: rendimiento, eficiencia... como otros que pueden requerir la participación humana: usabilidad, etc.

Para cumplir los requisitos requeridos y especificados se han seleccionado los siguientes casos:

- *The Arcade Game Maker (AGM) product line* [SEI 07a]. Es una línea de productos creada para facilitar el aprendizaje y la experimentación en las líneas de producto software. La línea de productos incluye tres juegos arcada. La arquitectura software de este producto está documentada y disponible en la web, por lo que es posible realizar una evaluación a nivel de arquitectura. Se han validado los atributos de calidad de rendimiento y usabilidad.
- *Línea de productos Calculadora Software*. Esta línea de productos ha sido creada específicamente para validar el método propuesto. La línea incluye calculadoras software con diferentes funcionalidades y que pueden ser desplegadas en móviles y PCs. Se ha implementado con AHEAD, por lo que es posible generar calculadoras concretas y medir los atributos de calidad. Se han validado los atributos de calidad de usabilidad, eficiencia y rendimiento.
- *Graph Product line*. La línea de productos grafo es una línea de productos publicada como un problema estándar para evaluar metodologías de línea de productos [Lopez-Herrejon 01] [Batory 05]. Esta línea de productos está implementada con AHEAD, por lo que es posible generar los productos y medir los atributos de calidad. Se han validado los atributos de calidad: tiempo de ejecución y mantenibilidad.

A continuación se resumen los aspectos comunes que permiten validar la generalidad así como las diferencias para abarcar un rango de situaciones. De este modo se asegura que los casos son representativos.

Aspectos comunes:

- El tamaño de los casos de estudio es homogéneo (número de características similar) así como el número de atributos que se evalúa en todos los casos. Esto permite comparar los resultados de los casos de estudio y

sacar conclusiones acerca de las posibles causas de las diferencias en las reducciones de esfuerzo logradas.

- Son líneas de productos con un nivel de variabilidad no muy elevado pero lo suficientemente representativo para que se den situaciones de interacciones entre las características.

Aspectos variantes:

- Nivel de abstracción en la que se ha realizado la validación (diseño e implementación). En dos de los casos de estudio se han generado productos y se han medido los atributos de calidad mediante la ejecución de los mismos. Mientras que en otro de los casos, se ha evaluado la arquitectura software (a nivel de diseño).
- Aplicación del método durante el desarrollo de la línea de productos (desarrollo y aplicación del método en paralelo) o aplicación del método una vez la línea está desarrollada (ingeniería inversa para obtener los modelos). La línea de productos Calculadora Software ha sido desarrollada mientras se aplicaba el método frente a la *Graph Product Line* que ya estaba codificada.
- Los atributos evaluados difieren de un caso a otro, así como su naturaleza. En cada caso, se han medido o evaluado aspectos diferentes: en algunos casos medibles más fácilmente por su naturaleza (por ejemplo aspectos de tiempo) y en otros casos más subjetivos de medir como la usabilidad. Los atributos que se han evaluado son los siguientes: usabilidad, rendimiento, mantenibilidad y eficiencia. La mayoría de los atributos son operacionales pero también se ha seleccionado uno que es de desarrollo (la mantenibilidad).
- El número de dependencias del modelo de características varía de un caso a otro. Esto ha permitido validar el método con modelos de características de propiedades diferentes.

En los casos de estudio se ha medido el ahorro que representa utilizar el método para validar la línea en vez de validar los productos uno a uno. Para ello se ha utilizado un doble enfoque: *top down* (utilizando el método) y evaluando cierto número de productos y *bottom up* (generando todos los productos o diseños) y evaluando o midiendo dichos productos. De este modo es posible comparar por un lado el esfuerzo y tiempo que se ha dedicado en un caso y en el otro, así como comprobar si se obtienen los mismos resultados o similares. Si se comprueba que los resultados son iguales o muy similares y que el primer enfoque requiere menos esfuerzo, es una forma de validar que el método desarrollado permite evaluar los atributos de calidad de una forma más eficiente (*cost-effective*).

7.2. Descripción de los casos de estudio

En las secciones siguientes se presenta un pequeño resumen de los casos de estudio que se encuentran detallados en los anexos G, H y I. Cada caso de estudio se presenta con el objetivo, modelo de características y una tabla resumen de los resultados de dicho caso de estudio.

7.2.1. Caso: Línea de Productos Juegos Arcade

El objetivo a la hora de aplicar el método en este caso de estudio ha sido validar la reducción de costes aplicando el método propuesto realizando una evaluación de la arquitectura software de la línea.

La línea de productos pedagógica *The Arcade Game Maker* (AGM) desarrollado por SEI (*Software Engineering Institute*) [SEI 07a] es una línea de productos creada para facilitar el aprendizaje y la experimentación en las líneas de producto software. La línea de productos incluye tres juegos arcada diferentes (*Brickle* o Ladrillos, Pong y Bolos). Todos los productos tienen en común el uso de un conjunto de elementos gráficos o *sprites* (móviles y estacionarios), animación gráfica y un conjunto de reglas acerca de la interacción de las piezas del juego. Sin embargo, el tipo de elemento (disco, remo, bola de bolos, ladrillo...) y el número es variante dependiendo del juego, así como las reglas, el comportamiento de los elementos y del entorno (ver figura 7.1).

Características evaluación				Resultados			
Atributos evaluados	Fase de evaluación	Estrategia de reducción	Técnicas de evaluación	N° total productos	N° productos evaluados	Reducción esfuerzo	Grado de confianza
Usabilidad Rendimiento	Diseño	Generación de un modelo genérico	Método de evaluación de arquitecturas software	32	Todos	Alto: el esfuerzo de especificar un modelo genérico es pequeño frente a evaluar todas los productos uno a uno.	Muy alto: Todos los productos se evalúan por lo que se obtienen los mismos resultados que evaluando uno a uno.

Obtener un modelo genérico de evaluación facilita mucho la evaluación de la línea. El esfuerzo de especificar el modelo genérico es pequeño comparado con el esfuerzo y coste de evaluar los productos uno a uno.

En este ejemplo en particular, el grado y número de interacciones es bastante alto debido a la naturaleza del atributo de calidad que se evalúa pero el modelo genérico permite evaluar todos los productos de forma sencilla y detectar las

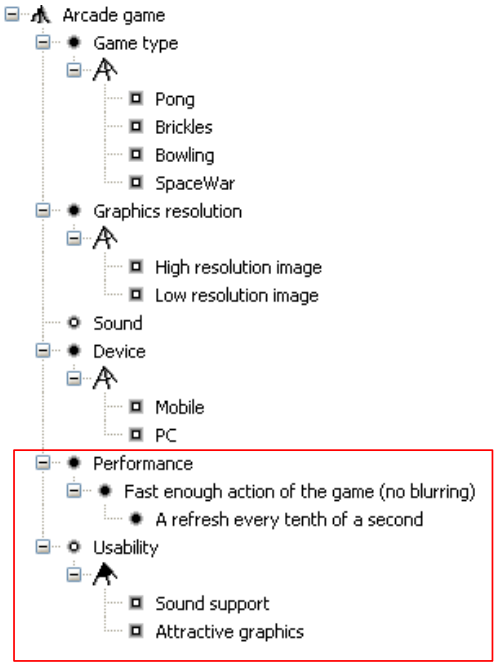


Figura 7.1: Modelo CaFM de la LP Arcade

interacciones existentes.

7.2.2. Caso: Línea de Productos Calculadora Software

El objetivo a la hora de aplicar el método en este caso de estudio ha sido validar la reducción de costes aplicando el método propuesto, en concreto la estrategia de selección de un subconjunto de productos realizando una validación de los atributos mediante la ejecución y medición de los atributos de calidad.

La línea de productos Calculadora software ha sido desarrollado específicamente para utilizarla durante la validación del método. La línea de productos se ha codificado utilizando AHEAD *Tool Suite* [Batory 04]. En AHEAD la derivación de los productos se puede realizar de forma automática de manera que obtener los productos de la línea es una tarea sencilla. La línea de productos incluye calculadoras para dispositivos con diferentes niveles de capacidad (Móviles, PCs...) y diferentes funcionalidades (ver figura 7.2).

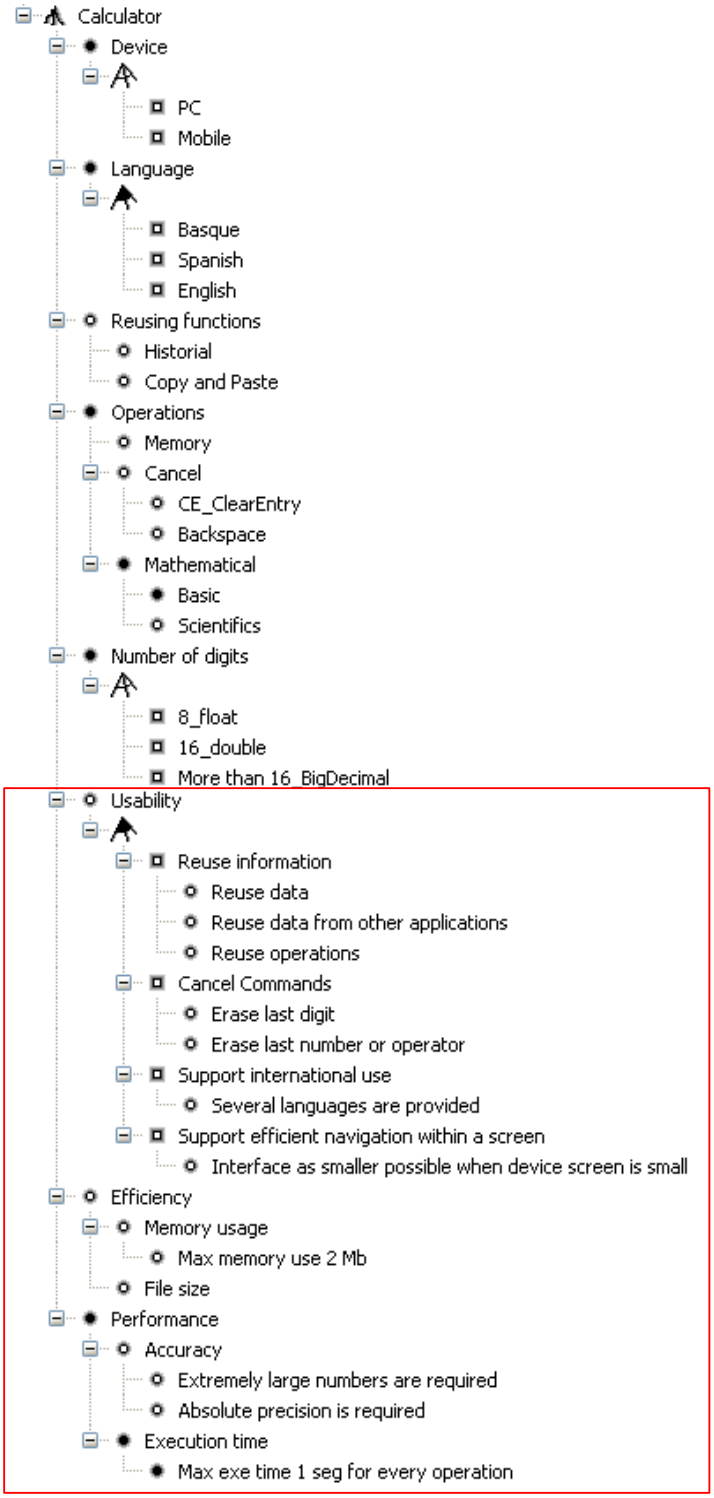


Figura 7.2: Modelo CaFM de la LP calculadora

Características evaluación				Resultados			
Atributos evaluados	Fase de evaluación	Estrategia de reducción	Técnicas de evaluación	N° total productos	N° productos evaluados	Reducción esfuerzo	Grado de confianza
Usabilidad Rendimiento Eficiencia	Implementación	Selección de un subconjunto de productos	Medición de los aspectos de calidad mediante ejecución de los productos	2688	50	98 %	98 %

En total se evalúan 50 productos de 2688 (Evaluando el 1,86 % de los productos es suficiente para detectar y cuantificar los impactos detectados). El porcentaje de los productos a evaluar es muy pequeño para evaluar toda la línea, lo que demuestra que aplicando el método se reduce considerablemente el esfuerzo y coste de validación.

7.2.3. Caso: GPL (Graph Product line)

El objetivo a la hora de aplicar el método en este caso de estudio ha sido validar la reducción de costes aplicando el método propuesto, en concreto la estrategia de selección de un subconjunto de productos realizando una validación de los atributos mediante la ejecución y medición de los atributos de calidad, en este caso no sólo atributos de calidad operacionales sino también de desarrollo.

La línea de productos grafo es una línea de productos que fue publicada como un problema estándar para evaluar metodologías de línea de productos [Lopez-Herrejon 01] [Batory 05]. Se ha trabajado con esta línea de productos implementada con AHEAD. Esta línea de productos genera aplicaciones para trabajar con grafos (ver figura 7.3), las aplicaciones pueden trabajar con uno o más algoritmos de grafos.

Características evaluación				Resultados			
Atributos evaluados	Fase de evaluación	Estrategia de reducción	Técnicas de evaluación	N° total productos	N° productos evaluados	Reducción esfuerzo	Grado de confianza
Rendimiento Mantenibilidad	Implementación	Selección de un subconjunto de productos	Medición de los aspectos de calidad mediante ejecución de los productos	44	160	72,5 %	Alto

El método está orientado principalmente a evaluar atributos de calidad operacionales que pueden ser variantes de un producto a otro pero también se puede utilizar para atributos de calidad de desarrollo importantes en una línea de productos tales como la mantenibilidad. El modelo de características extendido también puede ser utilizado para mejorar la calidad a nivel de línea de productos. En este caso, los valores de *maintainability index* de los productos pueden ayudan a identificar las variantes que disminuyen la mantenibilidad. Esta infor-

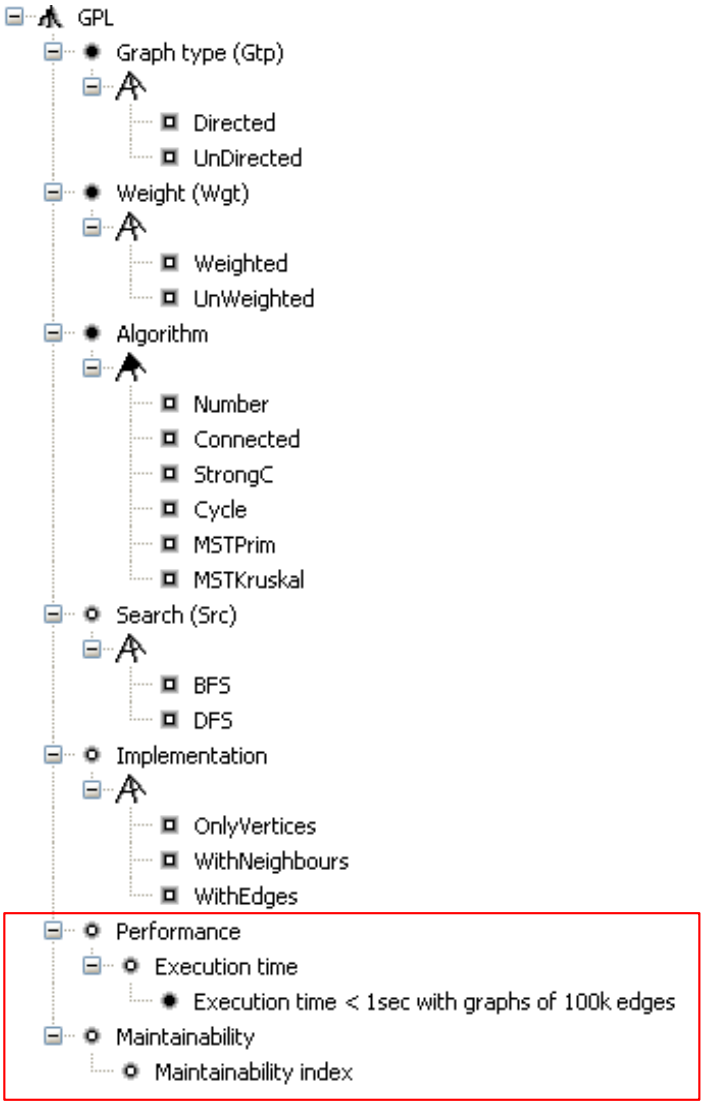


Figura 7.3: Modelo CaFM del GPL

mación es muy útil durante el mantenimiento preventivo para centrarse en esas características y mejorar su *maintainability index* mediante refactorización o para obtener el nivel de mantenibilidad de los productos durante la derivación.

El análisis de los resultados de las mediciones ayuda a corregir asunciones incorrectas realizadas por los diseñadores a la hora de especificar los impactos. En este caso, se suponía que un diseño podría ofrecer mejores resultados de tiempo con algunos algoritmos pero la experimentación ha demostrado que no es así al menos para los tamaños de grafos con las que se ha probado.

En total se han evaluado 44 productos de un total de 160 (27,5 % del total de productos). En este caso de estudio el número de productos a evaluar es superior que en el resto de los casos de estudio porque todas las características afectan a los atributos de calidad elegidos para evaluar.

7.3. Conclusiones globales de los casos de estudio

Como se muestra en la tabla 7.1, las conclusiones sobre la validación de la reducción del trabajo a la hora de validar son muy positivas. En todos los casos de estudio, se ha reducido el esfuerzo de validación considerablemente (frente a validar todos los productos). Esta reducción ha sido más considerable en algunos casos dependiendo del caso de estudio (número de dependencias...) y del atributo de calidad a evaluar (número de impactos...). En el caso de la selección de un subconjunto de productos, el número de dependencias y el número de impactos incrementan el número de productos a evaluar.

Además, se han observado otras ventajas que proporciona el método:

- El método permite especificar y caracterizar los atributos de calidad a nivel de requisitos. Algo muy interesante aunque no se vaya a evaluar o validar la calidad.
- El método permite tener en mente los aspectos de calidad durante el diseño e implementación y capturar el razonamiento de algunas decisiones.
- El método facilita la derivación con los atributos de calidad en mente.

Por la naturaleza de los atributos de calidad es frecuente que existan interacciones y esto aumenta el número de evaluaciones a realizar. En ese caso, es mejor

Tabla 7.1: Resumen de los casos de estudio

Caso de estudio	**Atributos de calidad**	**Estrategia de reducción de esfuerzo de evaluación**	**Técnicas de evaluación utilizadas**	**Conclusiones**
Arcade	Rendimiento, Usabilidad	Modelo de evaluación genérico	Evaluación de Arquitectura Software	Considerable reducción del coste de evaluación
Calculadora	Usabilidad, Eficiencia, Rendimiento	Selección de productos	Medición de la calidad mediante ejecución de productos	Un 98 % de reducción en el coste de evaluación
GPL	Tiempo de ejecución, Mantenibilidad	Selección de productos	Medición de la calidad mediante ejecución de productos	Un 72,5 % de reducción en el coste de evaluación

seguir la estrategia de desarrollar un modelo genérico que permita evaluar incluso todos los productos si resulta necesario pero con un coste bajo. Además, según la exactitud deseada se pueden obviar ciertas interacciones. El método ayuda a hacer que las interacciones entre características sean explícitas, algo importante y que no se suele considerar.

El método está pensado para métodos de evaluación cuantitativos ya que el objetivo es evaluar aspectos de calidad operacionales y normalmente son medidos y evaluados con métodos cuantitativos. Sin embargo, también es posible utilizar métodos cualitativos si hay una labor de cuantificación previa.

Capítulo 8

Conclusiones y líneas futuras

8.1. Conclusiones

Técnicas de modelado y evaluación existentes

Tras el estudio que se ha realizado de las técnicas y métodos de evaluación se puede concluir que hay pocos métodos para evaluar atributos de calidad operacionales en líneas de productos software y los que existen, se centran en atributos de calidad específicos y no existen métodos para todos los atributos de calidad. Por lo tanto, para la evaluación de atributos de calidad hay que recurrir a métodos que no son específicos para las líneas de productos, lo que hace que la evaluación sea más difícil ya que estos métodos no soportan la variabilidad.

Otro de los aspectos a considerar es el modelado de la variabilidad en los atributos de calidad, un aspecto clave para evaluar la calidad de una línea, ya que diferentes miembros de la línea pueden requerir diferentes niveles de calidad. La variabilidad en los atributos de calidad puede ser causada por la variabilidad funcional, por lo que esta relación debe quedar recogida a la hora de modelar la variabilidad en los atributos de calidad. Analizando los métodos y técnicas existentes que permiten modelar la variabilidad en los atributos de calidad, se llega a la conclusión de que a pesar de que existen varios enfoques o métodos, ninguno de ellos cumple todos los requisitos que se consideran necesarios.

Uno de los requisitos para el que menos soporte existe y tradicionalmente menos se ha tenido en cuenta es la necesaria modelización de la interacción

de los aspectos funcionales cuando estos aspectos impactan en la calidad. Las relaciones entre la variabilidad funcional, arquitectural y de implementación con la variabilidad en los atributos de calidad no siempre es una relación uno a uno, y cuando hay varias características que afectan a una misma característica de calidad, puede existir una interacción entre las características que implique un impacto diferente a la suma de los impactos individuales.

Otro aspecto que pocos métodos de modelización de variabilidad consideran en la actualidad es la naturaleza de los atributos de calidad. A pesar de existir varias clasificaciones y definiciones de atributos de calidad, en la práctica los atributos de calidad a menudo tienen diferentes significados dependiendo del dominio. Por lo tanto, es necesario describir o caracterizar los atributos de calidad para que se exprese lo que significan en el dominio en el que se trabaja.

Contribución

Se ha realizado un estudio de la evaluación de atributos de calidad en las líneas de productos (sobre todo de la evaluación de la arquitectura), identificando los retos asociados, así como nuevos momentos de evaluación y proponiendo una clasificación de atributos de calidad en líneas de productos. También se ha realizado un estudio y comparativa de los métodos y métricas de evaluación específicos de las líneas de productos.

Otra de las áreas que se ha estudiado es la de la variabilidad en los atributos de calidad; los enfoques que permiten modelar esta variabilidad han sido comparados y se han identificado los requisitos que debe cumplir el modelado de la variabilidad en los atributos de calidad. Dentro de la variabilidad en los atributos de calidad, se han estudiado especialmente los impactos e interacciones entre características y se ha desarrollado un algoritmo que permite detectar las interacciones y cuantificar los impactos.

Teniendo en cuenta las limitaciones de los enfoques de especificación de variabilidad en los atributos de calidad, se ha propuesto una extensión al tradicional modelo de características. En el enfoque propuesto se han añadido todos los aspectos necesarios para cubrir todos los requisitos que se habían definido previamente.

Se ha definido un método que incluye un proceso que guía en la definición y uso del modelo de variabilidad propuesto junto a estrategias de reducción de esfuerzo en las evaluaciones. Las estrategias propuestas son la reutilización de modelos de evaluación y la reducción del alcance centrándose en los productos más significativos. Este método permite evaluar cualquier atributo de calidad (especialmente los atributos operacionales) utilizando métodos de

evaluación de arquitecturas o métodos de testeo existentes pero gestionando la variabilidad en los atributos de calidad y utilizando esta información para reducir esfuerzos de evaluación. El método también considera la derivación de productos; la especificación de la variabilidad en los atributos de calidad, y los resultados de las evaluaciones son utilizados para facilitar una derivación teniendo en cuenta los niveles de atributos de calidad.

También se han extendido herramientas existentes y se ha implementado el algoritmo de selección de productos y detección de interacciones para dar soporte al método propuesto.

Características del método propuesto

El método desarrollado tiene un enfoque generalista. No está orientado a un atributo de calidad en particular o a un modelo de ciclo de vida, sino que tiene como objetivo la evaluación de cualquier atributo de calidad, independientemente del dominio, modelo de ciclo de vida o método de evaluación.

El método sirve para cualquier atributo de calidad aunque está principalmente orientado a atributos de calidad operacionales o relevantes en un dominio, ya que éstos son los que pueden variar en los productos y se tiene que comprobar que todo el rango de posibles valores se cumple. No obstante, los atributos de calidad de desarrollo también se pueden evaluar utilizando el método definido. Normalmente estos atributos son importantes a nivel de línea de productos y no a nivel de productos; por ejemplo, la modificabilidad de un producto no es tan importante como la modificabilidad de la línea. Evaluar la modificabilidad de los productos y comprobar el rango de niveles de modificabilidad no siempre es relevante, pero en algunos casos sí es interesante y el método también puede servir para ello. Además, la evaluación de atributos de calidad de desarrollo puede servir para lograr información a nivel de producto trasladable a nivel de línea de productos que indique áreas de riesgo o mejora.

El método no está sujeto a definiciones concretas de atributos de calidad, puesto que las definiciones suelen variar mucho y ser específicas del dominio. Los atributos de calidad se definen y explican utilizando el modelo propuesto. De este modo, en cada caso los desarrolladores y *stakeholders* pueden especificar qué entienden por "seguridad", "rendimiento"... en el contexto de esa línea de productos concreta. Además, se pueden definir nuevos atributos de calidad que no aparezcan en las clasificaciones habituales siempre y cuando existan métodos para evaluarlos.

Al poder considerar cualquier atributo de calidad y definir nuevos, el método está abierto a cualquier dominio de aplicación: desde un dominio de aplica-

ciones empresariales donde los atributos de calidad son importantes pero no críticos, a dominios donde cumplir los atributos puede ser crítico (dominio de tiempo real, sistemas empotrados *safety-critical* donde incluso la vida de las personas depende del correcto funcionamiento del software, etc.)

El método está abierto a la mayoría de métodos de evaluación de arquitecturas o testeo. El enfoque que proponemos utiliza como base métodos ya existentes de evaluación de arquitecturas o testeo de atributos de calidad para sistemas únicos. Está pensado para métodos de evaluación cuantitativos, ya que el objetivo es evaluar aspectos de calidad operacionales y normalmente son medidos y evaluados con métodos cuantitativos. Sin embargo, también es posible utilizar métodos cualitativos si hay una labor de cuantificación previa. Para facilitar la selección de los métodos, se ha desarrollado una guía comparativa de métodos de evaluación de arquitecturas para sistemas únicos y atributos de calidad operacionales. Esta guía hace que el enfoque sea más aplicable, pues es más sencillo seleccionar el método más adecuado para cada caso. Sin embargo, el enfoque también se puede utilizar con otros métodos (no considerados en la comparativa) o futuros nuevos métodos que puedan surgir siempre y cuando sean métodos cuantitativos que permitan evaluar un atributo de calidad.

Respecto al momento de evaluación de la calidad, el enfoque considera dos momentos principales: durante el diseño mediante evaluación de la arquitectura o tras la implementación utilizando métodos de testeo o medición de atributos de calidad. Es aplicable a líneas de productos en desarrollo o ya implementadas. El enfoque se puede aplicar en paralelo con las fases y actividades del ciclo de vida de desarrollo de la línea, pero también se puede aplicar el método en una línea ya implementada, partiendo de los modelos y activos existentes si la línea ya está desarrollada.

El método es independiente de los enfoques de desarrollo y los modelos de ciclo de vida de desarrollo de la línea. Respecto a los enfoques de desarrollo, el método se puede utilizar con cualquier enfoque de desarrollo de líneas de productos: FOP (*Feature Oriented Programming*), Pulse, etc. Incluso se puede utilizar con enfoques que proponen otros modelos de variabilidad y no utilizan modelos de características. El modelo de características extendido propuesto es complementario a otros modelos de variabilidad a nivel de diseño, implementación o requisitos. El enfoque es independiente de la notación utilizada para describir los requisitos, el diseño o el código y su variabilidad, puesto que el modelo CaFM es complementario: es un modelo central de la variabilidad existente. En enfoques de desarrollo que ya proponen modelos de características: FODA, FORM, FOPLE, el enfoque de *Riebisch*, Forfamel, etc. habría que adaptar el uso de esos modelos para añadir los aspectos de extensión y adoptar un uso del modelo durante todo el ciclo de vida.

Para ilustrar el proceso propuesto, se ha integrado en conjunto con un modelo de ciclo de vida de desarrollo clásico (ver figura 4.1), pero las mismas actividades del proceso se pueden integrar en otros modelos de ciclo de vida.

Base del modelo y del método propuesto

El método se basa en el modelo de características, que es un estándar de facto para modelar la variabilidad en las líneas de productos. Las características son una forma muy eficiente de comunicación entre los clientes y los desarrolladores porque permiten abstraer conceptos y también son una forma muy intuitiva de representar aspectos variables y comunes de forma sencilla.

En concreto el modelo CaFM se basa en dos enfoques de modelado de características: FeatuRSEB y CBFM. FeatuRSEB se ha seleccionado principalmente por la filosofía de utilizar el modelo de características como un modelo central que recoge la variabilidad a lo largo de todo el desarrollo. Propone tres tipos de características: funcionales, arquitecturales y de implementación. De este modo, al principio se definen las características funcionales, luego se van añadiendo las arquitecturales durante la fase de diseño y en la fase de codificación se añaden las características de implementación. En cuanto a la notación gráfica y para extender el metamodelo, se ha partido de *Cardinality Based Feature Modelling*.

A priori, el modelo de características sirve también para modelar atributos de calidad; una característica puede ser un aspecto de calidad pero, debido a la naturaleza de los atributos de calidad, hay limitaciones. Por ello, el modelo ha sido extendido a fin de permitir la caracterización y definición de atributos de calidad, permitiendo de este modo una independencia respecto a dominios y atributos de calidad que se pueden evaluar. Para permitir esta caracterización se ha integrado una extensión del árbol de utilidad de atributos de calidad en el modelo de características. Este árbol es utilizado en las evaluaciones ATAM para describir y caracterizar los atributos de calidad para poder realizar evaluaciones. El árbol se ha utilizado con éxito durante muchos años y ofrece grandes ventajas que también son útiles a la hora de definir los requisitos de calidad. Extendiendo y combinando este árbol con el resto de características funcionales, arquitecturales y de implementación se pueden especificar las relaciones de variación indirecta como impactos entre características. Esta información es extremadamente útil durante la evaluación así como en la fase de diseño de la línea de productos.

La mezcla de la caracterización para evaluaciones con el modelo de características clásico da como resultado un modelo de características extendido que es útil en todas las fases de desarrollo. Durante la ingeniería de requisitos, ayu-

da a capturar y definir los atributos de calidad requeridos y su variabilidad. Durante el diseño permite tener en cuenta los aspectos de calidad, capturar el razonamiento de algunas decisiones, recoger las relaciones de impacto entre las características arquitecturales y los atributos de calidad y realizar una primera valoración de si los atributos de calidad se van a cumplir. Durante la implementación, permite también tener en mente los aspectos de calidad y capturar las relaciones de impacto. Por supuesto, es de vital importancia en la fase de evaluación de atributos de calidad a nivel de arquitectura o tras la implementación. La información caracterizada de los atributos de calidad y las relaciones entre las características sirven para guiar la evaluación. Partiendo de esta información se pueden aplicar estrategias de reducción de esfuerzos, como la construcción de un modelo genérico y/o la selección de los productos más significativos que permitan evaluar que el rango de niveles requerido se cumple. Tras la evaluación, el modelo de características extendido se puede aumentar con características de derivación y utilizar la información de la evaluación para cuantificar los impactos. De este modo, el modelo de características extendido se puede utilizar para facilitar una derivación teniendo en cuenta atributos de calidad.

Conclusiones de los casos de estudio

Las conclusiones sobre la validación de la reducción de la carga de trabajo a la hora de evaluar son muy positivas. En todos los casos de estudio, se ha reducido el esfuerzo de evaluación considerablemente (frente a evaluar todos los productos). Esta reducción ha sido más considerable en algunos casos dependiendo del caso de estudio, cuantas más dependencias e impactos más productos hay que evaluar. El caso en el que mejores resultados se obtienen es cuando es posible obtener un modelo de evaluación genérico que permita evaluar todos los productos con un esfuerzo mínimo. Cuando se opta por evaluar un subconjunto de productos, también se han obtenido muy buenos resultados: una reducciones del 72,5 % y 98 % del esfuerzo frente a evaluar todos los productos. El porcentaje de reducción del esfuerzo tiene una fuerte relación con el número y grado de las interacciones entre características cuando impactan en atributos de calidad evaluados.

8.2. Resultados

Durante la realización de la tesis, se han publicado parte de los resultados de la investigación en conferencias y revistas internacionales con el fin de validar

la dirección y resultados de la investigación. A continuación se detallan las publicaciones relacionadas con la tesis y los aspectos de la tesis que se presentan en las mismas.

- Leire Etxeberria, Goiuria Sagardui, "*Product-Line Architecture: New Issues for Evaluation*", In Obbink, H., Pohl, K. (ed): Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, Proceedings, LNCS 3714 Springer (2005), 174-185 [Etxeberria 05b]
  - Presenta un análisis de la evaluación de arquitecturas en líneas de productos: nuevos aspectos a considerar respecto a la evaluación de arquitecturas de un sólo sistema:
    - Clasificación de atributos de calidad en líneas de productos.
    - Nuevos momentos y técnicas de evaluación.
  - Estudio y comparativa de métodos y métricas de evaluación de arquitecturas de línea de productos.
- Leire Etxeberria, Goiuria Sagardui, "*Architectural Evaluation Framework for Product Lines*", In First International Conference on the Quality of Software Architectures, QoSA 2005, Erfurt, Germany, Proceedings Net.ObjectDays (2005), 505-512 [Etxeberria 05a]
  - Artículo de posición que describe un marco de evaluación de arquitecturas en líneas de productos.
- Leire Etxeberria, Goiuria Sagardui, Lorea Belategi, "*Modelling Variation in Quality Attributes*", First International Workshop on Variability Modelling of Software-Intensive Systems (VaMos 2007), Proceedings, (Eds.) Klaus Pohl, Patrick Heymans, Kyo-Chul Kang, Andreas Metzger, Lero Technical Report 2007-01, 2007 [Etxeberria 07]
  - Análisis de la variabilidad en los atributos de calidad: definición, requisitos para su correcta especificación, etc.
  - Estudio y comparativa de enfoques existentes para especificar variación en los atributos de calidad.
- Leire Etxeberria, Goiuria Sagardui, Lorea Belategi, "*Quality aware Software Product Line Engineering*", Journal of the Brazilian Computer Society (JBCS), Special Issue on Software Reuse, no. 1; Vol. 14; Mar. 2008 - ISSN 0104-6500 [Etxeberria 08d]
  - Análisis y propuesta de los aspectos a considerar para lograr cumplir los requisitos de calidad en líneas de productos software.
  - Análisis y comparativa de los métodos de modelado de variabilidad, diseño, evaluación de arquitecturas y testeo en líneas de productos.
- Leire Etxeberria, Goiuria Sagardui, "*Evaluation of Quality Attribute Va-*

riability in Software Product Families", 15th IEEE International Conference on The Engineering of Computer-Based Systems, ECBS 08, proceedings, pp 255-264, 2008 [Etxeberria 08a]

- Presenta el modelo para representar la variabilidad en los atributos de calidad.
- Resumen de la aplicación del método en la línea de productos *Arcade game maker*.

- Leire Etxeberria, Goiuria Sagardui, "*Quality Assessment in Software Product Lines*", H. Mei (Ed.): ICSR, 10th International Conference on Software Reuse, LNCS 5030, pp. 178-181, 2008 [Etxeberria 08b]
  - Presenta el proceso de aplicación del método para facilitar la evaluación de la calidad en las líneas de productos teniendo en cuenta la variabilidad en los atributos de calidad.
  - Se ilustra la aplicación del método con el caso de estudio línea de productos calculadora software.
- Leire Etxeberria, Goiuria Sagardui, "*Variability Driven Quality Evaluation in Software Product Lines*", aceptado para su publicación en 12th International Software Product Line Conference (SPLC 2008) [Etxeberria 08c]
  - Presenta las estrategias de reducción de esfuerzo del método CaLiPro para la evaluación de atributos de calidad.

8.3. Limitaciones

Durante la aplicación del método CaLiPro se propone utilizar un modelo de características extendido como modelo central de variabilidad. Otros modelos y artefactos (requisitos, diseño, código, etc.) de la línea también tienen variabilidad. Así que además de reflejar esa variabilidad en los modelos correspondientes, debe recogerse en el modelo que proponemos. Para lo que hay que mantener una trazabilidad entre el modelo CaFM y el resto de modelos y artefactos con variabilidad. La definición y gestión de dicha trazabilidad no se ha considerado en el método CaLiPro.

Si se desean evaluar muchos atributos de calidad de diversa naturaleza al mismo tiempo, el número de características que afectan a esos atributos puede resultar muy alto y por lo tanto, la selección de productos no produce una reducción considerable de esfuerzo. Del mismo modo, en sistemas muy gran-

des, la aplicación del método y el análisis de los resultados pueden resultar complejo. Se requiere una automatización mayor y una mejora de la herramienta desarrollada para dar soporte adecuado a líneas de productos software de mayor envergadura.

El método permite obtener información para realizar análisis de *trade-off* y detectar conflictos entre atributos de calidad. Sin embargo, no se proporciona una herramienta para facilitar esta tarea.

En dominios críticos y para ciertos atributos de calidad, la evaluación de todos los productos puede ser necesaria, bien por requisito de los estándares de certificación que hay que cumplir o bien por la naturaleza de las propiedades a analizar. En estos casos, la aplicación de métodos formales puede ser requerida para verificar y validar los aspectos de calidad. El método CaLiPro no contempla el uso de métodos formales aunque puede ser una técnica complementaria a estos para ayudar a detectar los productos más críticos y analizar esos productos en primer lugar.

8.4. Líneas futuras

Se está trabajando en la aplicación del método CaLiPro en un caso real en la industria y los primeros resultados parecen alentadores pero hay que seguir trabajando en el caso para obtener conclusiones definitivas.

Como líneas futuras, a corto plazo se va a mejorar la herramienta de soporte que se ha desarrollado. Actualmente la herramienta fmp-CaLiPro permite un soporte básico del proceso, pero se puede mejorar añadiendo aspectos gráficos, una mejor integración de las funcionalidades y añadiendo nuevas funcionalidades.

Una de las funcionalidades previstas es el análisis de los conflictos entre atributos de calidad. A partir del modelo y de los impactos definidos, se puede hacer un estudio de los conflictos entre atributos de calidad y mostrarlos de forma explícita.

Otro aspecto importante donde se requieren herramientas es durante la derivación. La herramienta guiCa ayuda durante esta fase, pero también se está trabajando en integrar la herramienta fmp-CaLiPro con el *framework* de razonamiento FAMA que permite utilizar técnicas automáticas de razonamiento

basadas en CSP (*Constraint Satisfaction Problems*). De este modo es posible realizar una derivación basada en restricciones y evitar añadir características que impidan lograr los niveles de calidad deseados.

El de los sistemas empotrados es uno de los dominios de aplicación más propicios para el método desarrollado, debido a que es muy común tener familias de productos con productos que requieren diferentes niveles de atributos de calidad. Los sistemas empotrados tienen un componente *hardware*, y este hardware también puede ser variable (diferentes microprocesadores, etc.) y afectar a los atributos de calidad. En nuestra opinión, la variabilidad en el hardware puede ser modelada de la misma forma que la variabilidad funcional, con impactos a los atributos de calidad. Sin embargo, un estudio de si se cubren todas las necesidades de la variabilidad hardware es necesario. Como línea futura se prevé realizar un caso de estudio con un sistema empotrado que tenga variabilidad en el hardware.

El método CaLiPro está principalmente orientado a evaluar líneas de productos en desarrollo o ya implementadas pero cuyos productos no han sido generados (al menos no todos). Sin embargo, el método también podría utilizarse durante la evolución y utilizar los resultados de posibles evaluaciones de productos generados para retroalimentar la base de información del método (el modelo CaFM y sobre todo los impactos) para un posterior uso. Para ello, hay que definir el proceso y soporte necesarios.

A medio plazo, y tomando como base el modelo de características extendido, una línea futura podría ser desarrollar un proceso y soporte para ayudar al diseño de una línea de productos teniendo en cuenta aspectos de calidad. Tal como se ha concluido, el modelo de características extendido es de gran utilidad durante el análisis de requisitos y diseño. Proporcionando unas guías y herramientas se podría mejorar la toma de decisiones durante el diseño, así como realizar una mejor captura del razonamiento o el porqué de las decisiones.

A largo plazo, y orientado a los sistemas empotrados críticos donde la seguridad y la fiabilidad son clave, el método y el modelo deberían poder integrarse con modelos que permiten especificar variabilidad en los atributos de calidad. En este sentido existen perfiles UML orientados a especificar atributos de calidad (QoS [Aagedal 02], MARTE [Faugere 07], etc.). Sin embargo, estos perfiles no tienen en cuenta la variabilidad que puede existir en los atributos. Hacen falta nuevos modelos y métodos de validación basados en modelos, que permitan describir modelos anotados con atributos de calidad y variabilidad en los mismos y validar la obtención de los atributos en una fase temprana.

Bibliografía

[Aagedal 02] Jan Øyvind Aagedal & Earl F. Ecklund Jr. *Modelling QoS: Towards a UML Profile.* In Jean-Marc Jézéquel, Heinrich Hussmann & Stephen Cook, editor, UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings, volume 2460 of *LNCS*, pages 275–289. Springer, 2002.

[Aldekoa 08] G. Aldekoa, S. Trujillo, G. Sagardui & O. Diaz. *Quantifying Maintainability in Feature Oriented Product Lines.* In 12th European Conference on Software Maintenance and Reengineering (CSMR). Athens, Greece. IEEE, April 2008.

[Alonso 98] Alejandro Alonso, Marisol García-Valls & Juan Antonio de la Puente. *Assessment of Timing Properties of Family Products.* In Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families, pages 161–169, London, UK, 1998. Springer-Verlag.

[America 05] Pierre America, Dieter K. Hammer, Mugurel T. Ionita, J. Henk Obbink & Eelco Rommes. *Scenario-based decision making for architectural variability in product families.* Software Process: Improvement and Practice, vol. 10, no. 2, pages 171–187, 2005.

[Antkiewicz 04] Michal Antkiewicz & Krzysztof Czarnecki. *FeaturePlugin: feature modeling plug-in for Eclipse.* In eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, pages 67–72, New York, NY, USA, 2004. ACM Press.

[Asikainen 03] Timo Asikainen, Timo Soininen & Tomi Männistö. *A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families.* In Frank van der Linden, editor, Software Product-Family Engineering, 5th International Workshop, PFE, Revised Papers, volume 3014 of *Lecture Notes in Computer Science*, pages 225–249. Springer, 2003.

[Asikainen 06] Timo Asikainen, Tomi Mannisto & Timo Soininen. *A Unified Conceptual Foundation for Feature Modelling.* In SPLC '06: Proceedings of the 10th International on Software Product Line Conference, pages 31–40, Washington, DC, USA, 2006. IEEE Computer Society.

[Atkinson 02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst & Jörg Zettel. Component-based product line engineering with uml. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Auerswald 02] Marko Auerswald, Martin Herrmann, Stefan Kowalewski & Vincent Schulte-Coerne. *Reliability-Oriented Product Line Engineering of Embedded Systems.* In PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering, pages 83–100, London, UK, 2002. Springer-Verlag.

[Babar 04] Muhammad Ali Babar & Ian Gorton. *Comparison of Scenario-Based Software Architecture Evaluation Methods.* In APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), pages 600–607, Washington, DC, USA, 2004. IEEE Computer Society.

[Babar 05] Muhammad Ali Babar, Xiaowen Wang & Ian Gorton. *Supporting Security Sensitive Architecture Design.* In Ralf Reussner, Johannes Mayer, Judith A. Stafford, Sven Overhage, Steffen Becker & Patrick J. Schroeder, editor, First International Conference on the Quality of Software Architectures, QoSA, Erfurt, Germany, Proceedings, volume 3712 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2005.

[Balsamo 01] S. Balsamo & M. Simeoni. *Deriving Performance Models from Software Architecture Specifications.* In European Simulation Multiconference 2001 (ESM), 2001.

[Balsamo 03a] S. Balsamo, A. D. Marco, P. Inverardi & M. Simeoni. *Software Performance State of the Art and Perspectives.* Report cs-2003-1, Dipartimento di In-

formatica, Unversita Ca'Foscari di Venezia, January 2003.

[Balsamo 03b] Simonetta Balsamo, Vittoria DeÑitto Personè & Paola Inverardi. *A review on queueing network models with finite capacity queues for software architectures performance prediction.* Perform. Eval., vol. 51, no. 2-4, pages 269–288, 2003.

[Barchini 05] G. E. Barchini. *Métodos de I+D de la Informática.* Revista Informática Educativa y Medios Audiovisuales, vol. 2, pages 16–24, 2005.

[Bass 98] Len Bass, Paul Clements & Rick Kazman. Software architecture in practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[Batory 04] Don S. Batory, JacobÑeal Sarvela & Axel Rauschmayer. *Scaling Step-Wise Refinement.* IEEE Trans. Software Eng., vol. 30, no. 6, pages 355–371, 2004.

[Batory 05] Don S. Batory. *Feature Models, Grammars, and Propositional Formulas.* In J. Henk Obbink & Klaus Pohl, editor, 9th International Conference on Software Product Lines, SPLC, Proceedings, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[Batory 08] Don Batory. *The AHEAD Tool Suite (ATS).* http://www.cs.utexas.edu/ schwartz/ATS.html, 2008.

[Bayer 99] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen & J.-M. DeBaud. *PuLSE: A Methodology to Develop Software Product Lines.* In SSR'99. Proceedings of the 5th Symposium on Software Reusability, pages 122–131, Los Angeles, CA, USA, may 1999. ACM.

[Becker 03] M. Becker. *Towards a General Model of Variability in Product Families.* In Proceedings of the 1st Workshop on Software Variability Management, 2003.

[Bednasch] T. Bednasch, C. Endler & M. Lang. *CaptainFeature.* Tool available on SourceForge at https: //sourceforge.net/projects/captainfeature/.

[Bednasch 02] T. Bednasch. *Konzept und Implementierung eines konfigurierbaren Metamodells für die Merkmalmo-*

dellierung. Diplomarbeit, Fachbereich Informatik, Fachhochschule Kaiserslautern, Standort Zweibrücken, Germany, Oct 2002.

[Benavides 05] David Benavides, Pablo Trinidad & Antonio Ruiz-Cortés. *Automated Reasoning on Feature Models.* In Oscar Pastor & João Falcão e Cunha, editor, Advanced Information Systems Engineering, 17th International Conference, CAiSE, Proceedings, volume 3520 of *LNCS*, pages 491–503. Springer, 2005.

[Benavides 06] David Benavides, Sergio Segura, Pablo Trinidad & Antonio Ruiz-Cortés. *A first step towards a framework for the automated analysis of feature models.* In 10th International Software Product Line Conference (SPLC). IEEE Computer Society, 2006.

[Benavides 07] D. Benavides, S. Segura, P. Trinidad & A. Ruiz-Cortés. *FAMA: Tooling a Framework for the Automated Analysis of Feature Models.* In Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS), 2007.

[Bengtsson 98] P. Bengtsson & J. Bosch. *Scenario-Based Software Architecture Reengineering.* In ICSR '98: Proceedings of the 5th International Conference on Software Reuse, page 308, Washington, DC, USA, 1998. IEEE Computer Society.

[Bengtsson 99] Olof Bengtsson & Jan Bosch. *Architecture Level Prediction of Software Maintenance.* In CSMR '99: Proceedings of the Third European Conference on Software Maintenance and Reengineering, page 139, Washington, DC, USA, 1999. IEEE Computer Society.

[Bengtsson 04] PerOlof Bengtsson, Nico Lassing, Jan Bosch & Hans van Vliet. *Architecture-level modifiability analysis (ALMA).* J. Syst. Softw., vol. 69, no. 1-2, pages 129–147, 2004.

[Bertolino 03] Antonia Bertolino & Stefania Gnesi. *PLUTO: A Test Methodology for Product Families.* In Software Product-Family Engineering, 5th International Workshop, PFE, Revised Papers, pages 181–197, 2003.

[Beuche 04] Danilo Beuche, Holger Papajewski & Wolfgang

Schröder-Preikschat. *Variability management with feature models.* Sci. Comput. Program., vol. 53, no. 3, pages 333–352, 2004.

[Bosch 00] Jan Bosch. Design and use of software architectures: adopting and evolving a product-line approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Capilla 02] Rafael Capilla & Juan C. Dueñas. *Modelling Variability with Features in Distributed Architectures.* In PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering, pages 319–329, London, UK, 2002. Springer-Verlag.

[Cechticky 04] V. Cechticky, A. Pasetti, O. Rohlik & W. Schaufelberger. *XML-Based Feature Modeling.* In J. Bosch & C. Krueger, editor, Software Reuse: Methods, Techniques, and Tools, 8th International Conference, ICSR. Springer-Verlag, 2004.

[Chastek 01] Gary Chastek, Patrick Donohoe, Kyo Chul Kang & Steffen Thiel. *Product Line Analysis: A Practical Introduction.* Technical Report CMU/SEI-2001-TR-001, SEI, June 2001.

[Cheung 80] R. C. Cheung. *A User-Oriented Software Reliability Model.* IEEE Trans. Softw. Eng., vol. 6, no. 2, pages 118–125, 1980.

[Chung 99a] Lawrence Chung, Daniel Gross & Eric S. K. Yu. *Architectural Design to Meet Stakeholder Requirements.* In Proceedings of the First Working IFIP Conference on Software Architecture (WICSA), pages 545–564, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

[Chung 99b] Lawrence Chung, Brian A. Nixon, Eric Yu & John Mylopoulos. Non-functional requirements in software engineering (the kluwer international series in software engineering volume 5). Springer, October 1999.

[Clauß 01] Matthias Clauß. *Modeling Variability with UML.* In Proceedings of GCSE2001. Young Researchers Workshop, 2001.

[Clements 01] Paul Clements & Linda Northrop. Software pro-

duct lines: Practices and patterns. Addison-Wesley Professional, August 2001.

[Clements 02] Paul Clements, Rick Kazman & Mark Klein. Evaluating software architectures: Methods and case studies. Addison-Wesley Professional, January 2002.

[COM 07] 31st annual international computer software and applications conference (compsac 2007), 24-27 july 2007, beijing, china. IEEE Computer Society, 2007.

[Cortellessa 02] Vittorio Cortellessa, Harshinder Singh & Bojan Cukic. *Early reliability assessment of UML based software models.* In WOSP '02: Proceedings of the 3rd international workshop on Software and performance, pages 302–309, New York, NY, USA, 2002. ACM.

[Czarnecki 98] Krzysztof Czarnecki. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models.* PhD thesis, Technische Universitlt Ilmenau, Germany, 1998.

[Czarnecki 00] Krzysztof Czarnecki & Ulrich Eisenecker. Generative programming: Methods, tools, and applications. Addison-Wesley Professional, June 2000.

[Czarnecki 02] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger & Ulrich W. Eisenecker. *Generative Programming for Embedded Software: An Industrial Experience Report.* In GPCE '02: Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering, pages 156–172, London, UK, 2002. Springer-Verlag.

[Czarnecki 04] Krzysztof Czarnecki, Simon Helsen & Ulrich W. Eisenecker. *Staged Configuration Using Feature Models.* In Robert L. Nord, editor, SPLC, volume 3154 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2004.

[Czarnecki 05a] Krzysztof Czarnecki, Michal Antkiewicz, Chang Hwan Peter Kim, Sean Lau & Krzysztof Pietroszek. *Model-driven software product lines.* In OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pages 126–127,

New York, NY, USA, 2005. ACM.

[Czarnecki 05b] Krzysztof Czarnecki, Simon Helsen & Ulrich W. Eisenecker. *Formalizing cardinality-based feature models and their specialization.* Software Process: Improvement and Practice, vol. 10, no. 1, pages 7–29, 2005.

[Dashofy 02] Eric M. Dashofy, André van der Hoek & RichardÑ. Taylor. *An infrastructure for the Rapid Development of XML-based Architecture Description Languages.* In Proceedings of ICSE, 2002.

[de Bruin 01] Hans de Bruin & Hans van Vliet. *Feature and Feature Interaction Modeling with Feature-Solution Graphs.* In Proceedings of the GCSE'01 Feature Modeling Workshop, 2001.

[Deelstra 04] Sybren Deelstra, Marco Sinnema, Jos Nijhuis & Jan Bosch. *COSVAM: A Technique for Assessing Software Variability in Software Product Families.* In ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance, pages 458–462, Washington, DC, USA, 2004. IEEE Computer Society.

[Deelstra 05] Sybren Deelstra, Marco Sinnema & Jan Bosch. *Product derivation in software product families: a case study.* J. Syst. Softw., vol. 74, no. 2, pages 173–194, 2005.

[Dobrica 02] Liliana Dobrica & Eila Niemelä. *A survey on software architecture analysis methods.* IEEE Trans. Softw. Eng., vol. 28, no. 7, pages 638–653, 2002.

[Dolan 01] Thomas J. Dolan. *Architecture Assessment of Information-System Families: a practical perspective.* PhD thesis, Tech. Univ. Eindhoven, Netherlands, 2001.

[Eriksson 05] Magnus Eriksson, Jürgen Börstler & Kjell Borg. *The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations.* In J. Henk Obbink & Klaus Pohl, editor, SPLC, volume 3714 of *Lecture Notes in Computer Science*, pages 33–44. Springer, 2005.

[Etxeberria 05a] Leire Etxeberria & Goiuria Sagardui. *Architectural Evaluation Framework for Product Lines.* In

In First International Conference on the Quality of Software Architectures, QoSA, Proceedings Net.ObjectDays, pages 505–512, 2005.

[Etxeberria 05b] Leire Etxeberria & Goiuria Sagardui. *Product-Line Architecture: New Issues for Evaluation.* In J. Henk Obbink & Klaus Pohl, editor, 9th International Conference on Software Product Lines, SPLC, Proceedings, volume 3714 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2005.

[Etxeberria 07] Leire Etxeberria, Goiuria Sagardui & Lorea Belategi. *Modelling Variation in Quality Attributes.* In Klauss Pohl, Pratrick Heymans, Kyo-Chul Kang & Andreas Metzger, editor, First International Workshop on Variability of Software-Intensive Systems (VaMos 2007), volume Lero Technical report 2007-1. Lero, 2007.

[Etxeberria 08a] Leire Etxeberria & Goiuria Sagardui. *Evaluation of Quality Attribute Variability in Software Product Families.* In 15th IEEE International Conference on Engineering of Computer-Based Systems, 2008.

[Etxeberria 08b] Leire Etxeberria & Goiuria Sagardui. *Quality Assessment in Software Product Lines.* In ICSR (International Conference on Software Reuse), 2008.

[Etxeberria 08c] Leire Etxeberria & Goiuria Sagardui. *Variability Driven Quality Evaluation in Software Product Lines.* In l, editor, Accpeted for publication in International Conference on Software Product Lines, SPLC, 2008.

[Etxeberria 08d] Leire Etxeberria, Goiuria Sagardui & Lorea Belategi. *Quality aware Software Product Line Engineering.* Journal of the Brazilian Computer Society (JBCS), vol. 14, no. 1, Mar 2008. ISSN 0104-6500.

[Faugere 07] Madeleine Faugere, Thimothee Bourbeau, Robert de Simone & Sebastien Gerard. *MARTE: Also an UML Profile for Modeling AADL Applications.* In ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), pages 359–364, Washington, DC, USA, 2007. IEEE Computer Society.

[Ferber 02a] Stefan Ferber, Jörgen Haag & Juha Savolainen. *Feature Interaction and Dependencies: Modeling*

Features for Reengineering a Legacy Product Line. In SPLC 2: Proceedings of the Second International Conference on Software Product Lines, pages 235–256, London, UK, 2002. Springer-Verlag.

[Ferber 02b] Stefan Ferber, Peter Heidl & Peter Lutz. *Reviewing Product Line Architectures: Experience Report of ATAM in an Automotive Context.* In PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering, Lecture Notes in Computer Science, pages 364–382, London, UK, 2002. Springer-Verlag.

[Folmer 03] E. Folmer, J. Gurp & J. Bosch. *Scenario-Based Assessment of Software Architecture Usability.* In Proceedings of Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction, ICSE, Portland, 2003.

[Gallagher 00] Brian P. Gallagher. *Using the Architecture Tradeoff Analysis Method to Evaluate a Reference Architecture: A Case Study.* Technical Report CMU/SEI-2000-TN-007, SEI, 2000.

[Gannod 00] Gerald C. Gannod & Robyn R. Lutz. *An approach to architectural analysis of product lines.* In ICSE '00: Proceedings of the 22nd international conference on Software engineering, pages 548–557, New York, NY, USA, 2000. ACM Press.

[Golden 05] E. Golden, B.E. John & L. Bass. *Quality vs. quantity: comparing evaluation methods in a usability-focused software architecture modification task.* In International Symposium on Empirical Software Engineering, 2005.

[Gomaa 04a] Hassan Gomaa. Designing software product lines with uml: From use cases to pattern-based software architectures. Addison Wesley, 2004.

[Gomaa 04b] Hassan Gomaa & Diana L. Webber. *Modeling Adaptive and Evolvable Software Product Lines Using the Variation Point Model.* In HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9, page 90268.3, Washington, DC, USA, 2004. IEEE Computer Society.

[González-Baixauli 04] Bruno González-Baixauli, Julio Cesar Sampaio

do Prado Leite & John Mylopoulos. *Visual Variability Analysis for Goal Models.* In 12th IEEE International Conference on Requirements Engineering (RE), pages 198–207. IEEE Computer Society, 2004.

[Goseva-Popstojanova 01] Katerina Goseva-Popstojanova & Kishor S. Trivedi. *Architecture-based approach to reliability assessment of software systems.* Perform. Eval., vol. 45, no. 2-3, pages 179–204, 2001.

[Graaf 05] Bas Graaf, Hylke van Dijk & Arie van Deursen. *Evaluating an Embedded Software Reference Architecture: Industrial Experience Report.* In CSMR '05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, pages 354–363, Washington, DC, USA, 2005. IEEE Computer Society.

[Grassi 04] V. Grassi. *Architecture-based dependability prediction for service-oriented computing.* In Proceedings of the Twin Workshops on Architecting Dependable Systems, International Conference on Software Engineering (ICSE). Springer, 2004.

[Griss 98] M. Griss, J. Favaro & M. d'Alessandro. *Integrating Feature Modeling with the RSEB.* In 5th International Conference on Software Reuse, ICSR 1998, pages 76–85, Vancouver, BC, Canada, jun 1998.

[Gurp 01] Jilles Van Gurp, Jan Bosch & Mikael Svahnberg. *On the Notion of Variability in Software Product Lines.* In WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), Washington, DC, USA, 2001. IEEE Computer Society.

[Halmans 03] Günter Halmans & Klaus Pohl. *Communicating the Variability of a Software-Product Family to Customers.* Software and System Modeling, vol. 2, no. 1, pages 15–36, 2003.

[Henttonen 07] K. Henttonen, M. Matinlassi, E.Ñiemela & T. Kanstren. *Integrability and Extensibility Evaluation from Software Architectural Models - A Case Study.* The Open Software Engineering Journal, vol. 1, pages 1–20, 2007.

[Herzog 01] Ulrich Herzog & Jerome A. Rolia. *Performance va-*

lidation tools for software/hardware systems. Perform. Eval., vol. 45, no. 2-3, pages 125–146, 2001.

[Hotz 06] L. Hotz, K. Wolter & T. Krebs. Configuration in industrial product families: The conipf methodology. IOS Press, Inc., 2006.

[IEEE 90] IEEE. *IEEE standard glossary of software engineering terminology.* Technical report, 1990.

[IEEE 93] IEEE. *IEEE Standard 1061-1992. IEEE standard for a software quality metrics methodology*, 1993.

[IEEE 00] IEEE. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.* IEEE Std 1472000, 2000.

[Immonen 06] Anne Immonen. Software product lines, research issues in engineering and management, chapter A Method for Predicting Reliability and Availability at the Architecture Level, pages 373–422. Springer, 2006.

[Immonen 08] Anne Immonen & Eila Niemelä. *Survey of reliability and availability prediction methods from the viewpoint of software architecture.* Softw Syst Model, vol. 7, pages 49–65, 2008.

[Ionita 02] Mugurel T. Ionita, Dieter K. Hammer & Henk Obbink. *Scenario-Based Software Architecture Evaluation.* In Methods: An Overview, Workshop on Methods and Techniques for Software Architecture Review and Assessment at the International Conference on Software Engineering, Orlando, Florida, USA, May 2002.

[ISO/IEC 99] ISO/IEC. *Information Technology -Software Product Evaluation - Part 1: General overview.* ISO/IEC 14598-1, 1999.

[ISO/IEC 01] ISO/IEC. *Software engineering – Product quality – Part 1: Quality model.* ISO/IEC 9126-1:2001, 2001.

[Jansen 04] Anton Jansen, Rein Smedinga, Jilles van Gurp & Jan Bosch. *First class feature abstractions for product derivation.* IEE Proceedings - Software, vol. 151, no. 4, pages 187–198, 2004.

[Jarzabek 06] S. Jarzabek, B. Yang & S. Yoeun. *Addressing quality attributes in domain analysis for product lines.* IEE Proceedings - Software, vol. 153, no. 2, pages

61–73, 2006.

[Kang 90] K. Kang, S. Cohen, J. Hess, W.Ñovak & S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21, November 1990.

[Kang 98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin & Moonhang Huh. *FORM: A feature-oriented reuse method with domain-specific reference architectures.* Ann. Softw. Eng., vol. 5, pages 143–168, 1998.

[Kang 02] Kyo C. Kang, Jaejoon Lee & Patrick Donohoe. *Feature-Oriented Product Line Engineering.* IEEE Software, vol. 19, no. 4, pages 58–65, July/August 2002.

[Kauppinen 03] Raine Kauppinen & Juha Taina. *RITA Environment for Testing Framework-based Software Product Lines.* In Pekka Kilpeläinen & Niina Päivinen, editor, Proceedings of the Eighth Symposium on Programming Languages and Software Tools (SPLST), pages 58–69. University of Kuopio, Department of Computer Science, 2003.

[Kawauchi 03] S. Kawauchi & T.Ohta. *Modeling and Simulation for Detecting 3-way Feature Interactions in Telecommunication Services.* In Proc. of ATS03, 2003.

[Kazman 96] Rick Kazman, Gregory Abowd, Len Bass & Paul Clements. *Scenario-Based Analysis of Software Architecture.* IEEE Software, vol. 13, No. 6, pages 47–55, 1996.

[Klein 93] Mark Klein, Thomas Ralya, Bill Pollak, Ray Obenza & Michael González Harbour. A practitioner's handbook for real-time analysis: Guide to rate monotonic analysis for real-time systems. Kluwer Academic, 1993.

[Kolb 04] Ronny Kolb, John D. McGregor & Dirk Muthig, editor. First international workshop on quality assurance in reuse contexts (quarc), IESE-Report No. 096.04/E. Fraunhofer IESE, August 2004.

[Kolb 05] Ronny Kolb & Dirk Muthig, editor. First eworkshop on quality assurance for software product lines: Strategic issues, IESE-Report No. 013.05/E. Fraun-

hofer IESE, January 2005.

[Korhonen 04] Mika Korhonen & Tommi Mikkonen. *Assessing systems adaptability to a product family.* J. Syst. Archit., vol. 50, no. 7, pages 383–392, 2004.

[Krueger 02] Charles W. Krueger. *Variation Management for Software Production Lines.* In SPLC 2: Proceedings of the Second International Conference on Software Product Lines, pages 37–48, London, UK, 2002. Springer-Verlag.

[Kuusela 00] Juha Kuusela & Juha Savolainen. *Requirements engineering for product families.* In ICSE '00: Proceedings of the 22nd international conference on Software engineering, pages 61–69, New York, NY, USA, 2000. ACM Press.

[Lassing 99] Nico Lassing, Daan Rijsenbrij & Hans van Vliet. *Towards a Broader View on Software Architecture Analysis of Flexibility.* In APSEC '99: Proceedings of the Sixth Asia Pacific Software Engineering Conference, page 238, Washington, DC, USA, 1999. IEEE Computer Society.

[Lee 02] Kwanwoo Lee, Kyo Chul Kang & Jaejoon Lee. *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering.* In ICSR-7: Proceedings of the 7th International Conference on Software Reuse, pages 62–77, London, UK, 2002. Springer-Verlag.

[Lee 04] Kwanwoo Lee & Kyo Chul Kang. *Feature Dependency Analysis for Product Line Component Design.* In ICSR, pages 69–85, 2004.

[Liu 05] Jia Liu, Don S. Batory & Srinivas Nedunuri. *Modeling Interactions in Feature Oriented Software Designs.* In Feature Interactions in Telecommunications and Software Systems VIII, ICFI'05, 28-30 June 2005, Leicester, UK, pages 178–197, 2005.

[Lopez-Herrejon 01] Roberto E. Lopez-Herrejon & Don S. Batory. *A Standard Problem for Evaluating Product-Line Methodologies.* In GCSE '01: Proceedings of the Third International Conference on Generative and Component-Based Software Engineering, pages 10–24, London, UK, 2001. Springer-Verlag.

[Lopez-Herrejon 06] Roberto E. Lopez-Herrejon & Don S. Batory. *Modeling Features in Aspect-Based Product Lines with Use Case Slices: An Exploratory Case Study.* In Thomas KÃ$\frac{1}{4}$hne, editor, Models in Software Engineering, Workshops and Symposia at MoDELS, volume 4364 of *Lecture Notes in Computer Science*, pages 6–16. Springer, 2006.

[Lung 97] Chung-Horng Lung, Sonia Bot, Kalai Kalaichelvan & Rick Kazman. *An approach to software architecture analysis for evolution and reusability.* In CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research, page 15. IBM Press, 1997.

[Maccari 02] Alessandro Maccari. *Experiences in assessing product family software architecture for evolution.* In ICSE '02: Proceedings of the 24th International Conference on Software Engineering, pages 585–592, New York, NY, USA, 2002. ACM Press.

[Matinlassi 02] Mari Matinlassi, Eila Niemelä & Liliana Dobrica. *Quality-Driven Architecture Design and Quality Analysis Method: A Revolutionary Initiation Approach to a Product Line Architecture.* Technical Report VTT-PUBS-456, VTT Electronics, jan 2002.

[McGregor 01] John D. McGregor. *Testing a Software Product Line.* Technical Report CMU/SEI-2001-TR-022, SEI, dec 2001.

[Metzger 05] Andreas Metzger, Stan Bühne, Kim Lauenroth & Klaus Pohl. *Considering Feature Interactions in Product Lines: Towards the Automatic Derivation of Dependencies between Product Variants.* In Feature Interactions in Telecommunications and Software Systems VIII, ICFI'05, 28-30 June 2005, Leicester, UK, pages 198–216, 2005.

[Molter 99] G. Molter. *Integrating SAAM in Domain-Centric and Reuse-based Development Processes.* In Proc. of the 2nd Nordic Workshop on Software Architecture (NOSA), 1999.

[Myers 97] M. D. Myers. *Qulitative research in Information Systems.* MIS Quarterly, pages 241–242, June 1997.

[Myllärniemi 06] Varvana Myllärniemi, Tomi Männistö & Mikko

Raatikainen. *Quality Attribute Variability within a Software Product Family Architecture*. In 2nd International Conference on the Quality of Software Architectures (QoSA), 2006.

[Nebut 06] Clémentine Nebut, Yves Le Traon & Jean-Marc Jézéquel. Software product lines, research issues in engineering and management, chapter System Testing of Product Lines: From Requirements to Test Cases, pages 447–477. Springer, 2006.

[Niemelä 05a] Eila Niemelä. *Architecture Centric Software Family Engineering*. Product Family Engineering seminars, Helsinki, Finland, October 2005.

[Niemelä 05b] Eila Niemelä. *Quality Driven Family Architecture Development*. Tutorial in 9th International Conference on Software Product Lines, SPLC, 2005.

[Niemelä 07] Eila Niemelä & Anne Immonen. *Capturing quality requirements of product family architecture*. Inf. Softw. Technol., vol. 49, no. 11-12, pages 1107–1120, 2007.

[Olumofin 05] Femi G. Olumofin & Vojislav B. Misic. *Extending the ATAM Architecture Evaluation to Product Line Architectures*. In WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), pages 45–56, Washington, DC, USA, 2005. IEEE Computer Society.

[Olumofin 07] Femi G. Olumofin & Vojislav B. Mišić. *A holistic architecture assessment method for software product lines*. Inf. Softw. Technol., vol. 49, no. 4, pages 309–323, 2007.

[OMG 05] OMG. *Software Process Engineering Metamodel Specification, Version 1.1*. http://www.omg.org/docs/formal/05-01-06.pdf, January 2005.

[OMG 08] OMG. *UML Resource Page*. http://www.uml.org/, 2008.

[Pearse 95] T. Pearse & P. Oman. *Maintainability measurements on industrial source code maintenance activities*. In ICSM '95: Proceedings of the International Conference on Software Maintenance, page 295, Washington, DC, USA, 1995. IEEE Computer So-

ciety.

[Petriu 00] Dorina Petriu, Christiane Shousha & Anant Jalnapurkar. *Architecture-Based Performance Analysis Applied to a Telecommunication System.* IEEE Trans. Softw. Eng., vol. 26, no. 11, pages 1049–1065, 2000.

[Pohl 05] Klaus Pohl, Günter Böckle & Frank J. van der Linden. Software product line engineering : Foundations, principles and techniques. Springer, September 2005.

[Pohl 06] Klaus Pohl & Andreas Metzger. *Software product line testing.* Commun. ACM, vol. 49, no. 12, pages 78–81, 2006.

[POSTECH 08] POSTECH. *ASADAL/FORM CASE Tool.* http://selab.postech.ac.kr/form/, 2008.

[Prehofer 97] Christian Prehofer. *Feature-Oriented Programming: A Fresh Look at Objects.* In ECOOP, pages 419–443, 1997.

[Pulvermueller 02] Elke Pulvermueller, Andreas Speck, James Coplien, Maja D'Hondt & Wolfgang De Meuter. *Feature Interaction in Composed Systems.* In ECOOP '01: Proceedings of the Workshops on Object-Oriented Technology, pages 86–97, London, UK, 2002. Springer-Verlag.

[PureSystems 08] PureSystems. *Pure::variants tool.* http://www.pure-systems.com/, 2008.

[Purhonen 05] Anu Purhonen. *Performance Evaluation Approaches for Software Architects.* In Component-Based Software Development for Embedded Systems, volume 3778 of *Lecture Notes in Computer Science.* Springer, 2005.

[Rahman 04] Asim Rahman. Metrics for the structural assessment of product line architecture. Master's thesis, School of Engineering, Blekinge Institute of Technology, 2004.

[Reis 06] Sacha Reis, Andreas Metzger & klaus Pohl. *A Reuse Technique for Performance Testing of Software Product Lines.* In Peter Knauber, Charles Krueger & Tim Trew, editor, SPLIT 2006 - Third International Workshop on Software Product Line

Testing, volume Computer Science Reports. Mannheim University of Applied Sciences - Computer Science Department, 2006.

[Reussner 03] Ralf H. Reussner, Heinz W. Schmidt & Iman H. Poernomo. *Reliability prediction for component-based software architectures.* J. Syst. Softw., vol. 66, no. 3, pages 241–252, 2003.

[Riebisch 02] Matthias Riebisch, Kai Böllert, Detlef Streitferdt & Ilka Philippow. *Extending Feature Diagrams with UML Multiplicities.* Integrated Design and Process Technology, IDPT, June 2002.

[Riebisch 03] Matthias Riebisch. *Towards a More Precise Definition of Feature Models.* In M. Riebisch, J. O. Coplien & D. Streitferdt, editor, Modelling Variability for Object-Oriented Product Lines ECOOP Workshop, Darmstadt, Germany. BooksOnDemand Publ. Co., July 2003.

[Riva 03] Claudio Riva & Christian Del Rosso. *Experiences with Software Product Family Evolution.* In IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution, page 161, Washington, DC, USA, 2003. IEEE Computer Society.

[Rodrigues 05] GenaínaÑunes Rodrigues, David S. Rosenblum & Sebastián Uchitel. *Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems.* In Maura Cerioli, editor, FASE, volume 3442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2005.

[Roshandel 04] Roshanak Roshandel, André Van Der Hoek, Marija Mikic-Rakic & Nenad Medvidovic. *Mae—a system model and environment for managing architectural evolution.* ACM Trans. Softw. Eng. Methodol., vol. 13, no. 2, pages 240–276, 2004.

[Rosso 06] Christian Del Rosso. *Continuous evolution through software architecture evaluation: a case study: Practice Articles.* J. Softw. Maint. Evol., vol. 18, no. 5, pages 351–383, 2006.

[Schobbens 06] Pierre-Yves Schobbens, Patrick Heymans & Jean-Christophe Trigaux. *Feature Diagrams: A Survey and a Formal Semantics.* In RE '06: Proceedings of the 14th IEEE International Requirements En-

gineering Conference (RE'06), pages 136–145, Washington, DC, USA, 2006. IEEE Computer Society.

[SEI 07a] SEI. *Arcade Game Maker Pedagogical Product Line.* Arcade Game Maker Pedagogical Product Line. Web page: http://www.sei.cmu.edu/productlines/ppl/index.html, 2007.

[SEI 07b] SEI. *Maintainability Index Technique for Measuring Program Maintainability.* http://www.sei.cmu.edu/str/descriptions/mitmpm.html, 2007.

[SEI 07c] SEI. *Software Architecture Glossary (Software Engineering Institute, Carnegie Mellon University).* Web page: http://www.sei.cmu.edu/architecture/glossary.html, 2007.

[Sinnema 04] Marco Sinnema, Sybren Deelstra, Jos Nijhuis & Jan Bosch. *COVAMOF: A Framework for Modeling Variability in Software Product Families.* In Robert L. Nord, editor, 3rd International Conference on Software Product Lines, SPLC, Proceedings, volume 3154 of *LNCS*, pages 197–213. Springer, sep 2004.

[Sinnema 07] Marco Sinnema & Sybren Deelstra. *Classifying variability modeling techniques.* Inf. Softw. Technol., vol. 49, no. 7, pages 717–739, 2007.

[Smith 02] Connie U. Smith & Lloyd G. Williams. Performance solutions: a practical guide to creating responsive, scalable software. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

[Stafford 01] J.A. Stafford & K.C. Wallnau. *Predicting Feature Interactions in Component-Based Systems.* In Proceedings of the Workshop on Feature Interaction of Composed Systems, June 2001.

[Stoermer 03] Christoph Stoermer, Felix Bachmann & Chris Verhoef. *SACAM: The Software Architecture Comparison Analysis Method.* Technical Report CMU/SEI-2003-TR-006, SEI, 2003.

[Svahnberg 03] Mikael Svahnberg, Claes Wohlin, Lars Lundberg & Michael Mattsson. *A Quality-Driven Decision-Support Method for Identifying Software Architec-*

ture Candidates. International Journal of Software Engineering and Knowledge Engineering, vol. 13, no. 5, pages 547–573, 2003.

[Thiel 02] Steffen Thiel & Andreas Hein. *Systematic Integration of Variability into Product Line Architecture Design.* In SPLC 2: Proceedings of the Second International Conference on Software Product Lines, pages 130–153, London, UK, 2002. Springer-Verlag.

[Trujillo 07] Salvador Trujillo, Don Batory & Oscar Diaz. *Feature Oriented Model Driven Development: A Case Study for Portlets.* In ICSE '07: Proceedings of the 29th international conference on Software Engineering, pages 44–53, Washington, DC, USA, 2007. IEEE Computer Society.

[Turner 99] C. Reid Turner, Alfonso Fuggetta, Luigi Lavazza & Alexander L. Wolf. *A conceptual basis for feature engineering.* The Journal of Systems and Software, vol. 49, no. 1, pages 3–15, 1999.

[van~der Hoek 03] André van der Hoek, Ebru Dincel & Nenad Medvidovic. *Using Service Utilization Metrics to Assess the Structure of Product Line Architectures.* In METRICS '03: Proceedings of the 9th International Symposium on Software Metrics, page 298, Washington, DC, USA, 2003. IEEE Computer Society.

[van~der Hoek 04] André van der Hoek. *Design-time product line architectures for any-time variability.* Sci. Comput. Program., vol. 53, no. 3, pages 285–304, 2004.

[van~der Linden 04] Frank van der Linden, editor. Software product-family engineering, 5th international workshop, pfe 2003, siena, italy, november 4-6, 2003, revised papers, volume 3014 of *LNCS*. Springer, 2004.

[van Ommering 00] Rob van Ommering, Frank van der Linden, Jeff Kramer & Jeff Magee. *The Koala Component Model for Consumer Electronics Software.* Computer, vol. 33, no. 3, pages 78–85, 2000.

[von~der Maßen 03] Thomas von der Maßen & Horst Lichter. *RequiLine: A Requirements Engineering Tool for Software Product Lines.* In van der Linden [van~der Linden 04].

[Wang 99] Wen-Li Wang, Ye Wu & Mei-Hwa Chen. *An Architecture-Based Software Reliability Model.* In

PRDC '99: Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing, page 143, Washington, DC, USA, 1999. IEEE Computer Society.

[Weiss 99] David M. Weiss & Chi Tau Robert Lai. Software product-line engineering: A family-based software development process. AW, 1999.

[Weiss 05] M. Weiss & B. Esfandiari. *On Feature Interactions Among Web Services.* Int. J. Web Service Res., vol. 2, no. 4, pages 22–47, 2005.

[Wells 02] L. Wells. *Performance Analysis Using Coloured Petri Nets.* mascots, vol. 0, page 0217, 2002.

[Wijnstra 03] Jan Gerben Wijnstra. *Evolving a Product Family in a Changing Context.* In van der Linden [van˜der Linden 04], pages 111–128.

[Williams 02] Lloyd G. Williams & Connie U. Smith. *PASA: a method for the performance assessment of software architectures.* In WOSP '02: Proceedings of the 3rd international workshop on Software and performance, pages 179–189, New York, NY, USA, 2002. ACM Press.

[Yacoub 99] Sherif M. Yacoub, Bojan Cukic & Hany H. Ammar. *Scenario-Based Reliability Analysis of Component-Based Software.* In ISSRE '99: Proceedings of the 10th International Symposium on Software Reliability Engineering, page 22, Washington, DC, USA, 1999. IEEE Computer Society.

[Yin 94] Robert K. Yin. Case study research: Design and methods. Sage Publications, Inc, 1994.

[Yin 02] R. K. Yin. Case study research, design and methods. Newbury Park, Sage Publications, 2002.

[Zhang 03] Hongyu Zhang, Stan Jarzabek & Bo Yang. *Quality Prediction and Assessment for Product Lines.* In Johann Eder & Michele Missikoff, editor, 15th International Conference on Advanced Information Systems Engineering, CAiSE, Proceedings, volume 2681 of *LNCS*, pages 681–695. Springer, 2003.

[Zhang 04] Hongyu Zhang & Stan Jarzabek. *XVCL: a mechanism for handling variants in software product lines.* Sci. Comput. Program, vol. 53, pages 381–407,

2004.

[Zhuo 93] F. Zhuo, B. Lowther, P. Oman & J. Hagemeister. *Constructing and Testing Software Maintainability Assessment Models*. In First Int'l Software Metrics Symp.,Baltimore, Md., IEEE CS Press, 1993.

Glosario

Activo núcleo Un activo es un producto de trabajo con un valor potencial para la organización que puede ser usado (y reutilizado) en sus operaciones (desarrollo software). Los activos están diseñados para la reutilización, capturan la parte común del dominio y permiten la variabilidad para que se puedan adaptar a diferentes contextos. 15

ADL (Architecture Description Language) Lenguajes formal desarrollado específicamente para representar y analizar arquitecturas software. 15

Arquitectura Software La estructura o las estructuras que abarcan los componentes de software, las características externamente visibles de esos componentes y las relaciones entre ellas [Bass 98]. 2

Arquitectura software de línea de productos (ALP) Una arquitectura común para un conjunto de productos o sistemas relacionados desarrollados por una organización [Bosch 00]. 15

Atributo de calidad Una propiedad de un producto de trabajo o productos por los cuales su calidad es juzgada por algún stakeholder [SEI 07c] . 20

Calidad del software El grado en el que el software posee una combinación de atributos deseada [IEEE 93]. 1

Característica o *feature* Una propiedad de un sistema que es relevante para varios *stakeholders* y que es utilizado para capturar la parte común y discriminar entre sistemas [Czarnecki 04]. 48

Caso de estudio Una indagación empírica que investiga un fenómeno contemporáneo dentro de su contexto de vida real, especialmente cuando las fronteras entre el fenómeno y el contexto no son claramente evidentes [Yin 02]. 9

Derivación de productos El proceso completo de construir un producto a partir de los activos software de la línea de productos [Deelstra 05]. 4

Evaluación de la arquitectura El examen sistemático de la extensión con la que una arquitectura software cumple los requisitos [Dolan 01]. 2, 15

Evaluación de la calidad El examen sistemático de la extensión con la que una entidad es capaz de cumplir los requisitos especificados [ISO/IEC 99]. 44

Ingeniería de aplicacion Es el proceso de la ingeniería de línea de productos software en la cual las aplicaciones de la línea son construidas reutilizando los activos núcleo y explotando la variabilidad de la línea. 16

Ingeniería de dominio Es el proceso de la ingeniería de línea de productos software en la cual la parte común y variable de la línea de productos se define y se realiza. 16

Línea de Productos Software Conjunto de sistemas software intensivos que comparten un conjunto de características que satisfacen las necesidades específicas de un segmento de mercado particular y que son desarrollados a partir de un conjunto de activos núcleo en un modo preestablecido [Clements 01]. 1

Stakeholders Un *stakeholder* es un individuo, equipo u organización con intereses en, o precupaciones relativas, a un sistema [IEEE 00]. 1

UML (Unified Modelling Language) Notación estándar para modelar sistemas especialmente de software [OMG 08]. 17

Validación El proceso de asegurar que un producto o sistema cumple con los requisitos que tiene especificados. 44

Variabilidad La habilidad de cambiar o personalizar un sistema [Gurp 01]. 17

Anexo A

Conclusions and future work

A.1. Conclusions

Modelling and evaluation techniques[1]

After the existing investigating evaluation techniques and methods, it can be concluded that there are very few methods for evaluating operational quality attributes in software product lines. The few methods that are available focus only on the specific quality attributes so all the quality attributes are not covered. Therefore, methods that are not product line specific must be used for evaluating these quality attributes and as a result the evaluation is more difficult since these methods do not support the variability.

Another aspect to consider is the quality attribute variability, which is a key aspect to evaluate the quality of a line, because different members of the line can require different quality levels. The quality attribute variability can be caused by the functional variability; therefore this relationship must be captured when modelling the quality attribute variability. After analysing the available variability modelling techniques, it can be concluded that none of the approaches meet all the necessary requirements.

One of these requirements for which less support exists and it has been taking into account to a smaller degree is the modelling of the functional aspect interaction when those aspects impact on quality. The relationship between

[1]This appendix is a translation into English of the chapter 8 in order to meet the requirements to get the "doctor europeus" mention.

the functional, architectural and implementation variability, and the quality attribute variability is not always a one-to-one relationship, and when several features can have an effect on the same quality feature, an interaction can be formed between the features that could have a different impact on quality to the sum of all the impacts.

Another aspect that very few variability modelling methods consider is the nature of the quality attributes. Even if there are several classifications and definitions for the quality attributes, in practice the quality attributes can often have different meanings depending on the domain. Therefore, it is necessary to describe or characterise those quality attributes so they tell what they mean in the working domain.

Contribution

A study of the evaluation of quality attributes in product lines has been performed (especially of the architecture evaluation), the associated challenges have been identified, including the new evaluation moments, and a quality attribute classification specific for product lines has been proposed. An analysis and survey of the product line specific quality evaluation methods and metrics has also been performed.

Another of the investigated areas is that of the quality attribute variability; the approaches that allow the modelling of this variability have been compared and the requirements that must be met have been defined. Within this quality attribute variability, the impacts and interactions between features have been studied and an algorithm has been developed for detecting the feature interactions and quantify the impact on quality.

Taking into account the limitations of the variability specification approaches, a new model has been proposed. This model is an extension of another very used model: the feature model. In the proposed approach all the necessary aspects have been added to cover all the previously defined requirements.

A method has been defined that includes a process to guide during the specification and use of the proposed variability model as well as evaluation effort reduction strategies. The proposed strategies are the evaluation model reuse and the reduction of the evaluation scope focusing on the most significant products. This method allows the evaluation of any quality attribute (especially the operational ones) using existing architecture evaluation or testing methods but managing the quality attribute variability and using this information to reduce the evaluation effort. The product derivation is also considered in this method; the quality attribute variability specification and the evaluation

results are used to facilitate a quality attribute aware product derivation.

Existing tools have been extended and the product selection and the interaction detection algorithm have been implemented as a support for the proposed method.

Characteristics of the proposed method

The developed method has a broad point of view. It is not oriented towards a specific quality attribute or a life-cycle model, but its goal is to evaluate any quality attribute, independent of the domain, life-cycle model or evaluation method.

The method can be used for any quality attribute although it is mainly oriented to the operational or the domain relevant quality attributes, because those are the qualities that can vary in the products and all the range of possible values must be checked. Nevertheless, the development quality attribute can also be evaluated using the method. Usually these qualities are important at the product line level and not at the product level; for instance, the modifiability of a product is not so important as the modifiability of the line. To evaluate the modifiability of the products and to check the range of the modifiability levels is not always relevant, but in some cases it can be interesting and the aforementioned method can be used for doing that. Moreover, the evaluation of the development quality attributes can also be used to get information at the product level and this can be transferred to product line level to identify risk points or needed improvement areas.

The method is not subject to concrete quality attribute definitions, because those definitions can vary considerably and be domain specific. The quality attributes are defined and explained using the proposed model. This way, in each case the developers and stakeholders can specify what they mean by security, performance,... in their specific product line context. Moreover, new quality attributes that do not appear in the classical classifications can be defined as long as there are methods available to evaluate them.

As it is possible to consider any quality attribute and define new ones, the method is open to any application domain: from business applications where the quality attributes are important but not critical to domains where to meet these quality attributes is critical (real-time domain, safety-critical embedded systems where even the life of people depends on the correct behaviour of the software...)

The method is open to most of the architecture evaluation and testing methods.

The proposed approach is based on existing evaluation methods for single systems. It is intended for quantitative evaluation methods, because the goal is to evaluate operational quality attributes and usually those qualities are measured and evaluated with quantitative methods. However, it is also possible to use qualitative methods with a previous quantification work. To facilitate the method selection, a comparative guide of the architecture evaluation methods for single systems and the operational quality attributes has been developed. This guide makes this approach more applicable, since it is easier to select the most appropriate method for each case. Nevertheless, this approach can also be used with other methods (not considered in the present comparison) as well as any new method.

With respect to evaluation time, the approach considers two main moments: during the design stage using the architecture evaluation methods and after the implementation using the quality testing or measuring methods. The method can be applied to product lines that are in the development stage or already implemented. The approach can be applied in parallel with the phases and activities of the life-cycle development and the method can also be applied in an already implemented product line, starting from existing models and assets.

The method is independent of the product line development approaches and the life-cycle models. With respect to the development approaches, the method can be used with almost any product-line development approach: FOP (Feature Oriented Programming), Pulse, and so on. It can even be used with approaches that propose other variability models and they do not use feature models. The proposed extended feature model (CaFM) is complementary to other variability models at design, implementation or requirement level. The method is independent of the notation used to describe the requirements, the design or the code and their variability. Since the CaFM model is a complementary model that is a central model that gathers the existing variability. In the development approaches where the feature models are proposed: FODA, FORM, Generative Programming, and so on it is necessary to adopt the use of those models to add quality aspects and also to adopt the use of the model during all the cycle-life.

To illustrate the proposed process, it has been shown along with a classic life-cycle model (see figure 4.1), but the same process activities can be integrated in other life-cycle models.

Proposed model's and method's base

The method is based on the feature model, which is a standard de facto for modelling variability in software product lines. The features are a very efficient

way of communicating between customers and developers because they allow the extraction of concepts and they are also very useful for representing the variable and common aspects in a simple manner.

The CaFM model is based on two feature modelling approaches: FeatuRSEB and *Cardinality Based Feature Modelling*. The FeatuRSEB has been selected mainly for its philosophy of using the feature model as a central model that captures the variability along the whole development stage. Three types of features are proposed: the functional, the architectural and the implementation. This way, all the functional features are defined at the beginning, then architectural features are added at the design stage and at the implementation stage the implementation features are added. The graphical notation and the metamodel, has been based on the *Cardinality Based Feature Modelling*.

A priori, the feature model can be used for modelling the quality attributes; a feature can be a quality aspect but, due to the nature of the quality attributes, there are some limitations. For this reason, the feature model has been extended in order to allow the characterization and definition of the quality attributes; this way any quality attribute from any domain can be defined and evaluated. In order to allow this characterization an extension of the quality attribute utility tree has been integrated into the feature model. This tree is used in ATAM evaluations to describe and characterize the quality attributes in order to evaluate them. The tree has been used successfully for many years and it provides a lot of benefits that are useful when defining the quality requirements. Extending and combining this tree with the rest of the functional, architectural and implementation features, allows specifying the relationship of the indirect variations such as the impact among the features of the model. This information is extremely useful during the evaluation as well as during the design phase of the product line.

The combination of the characterization for the evaluation and the classic feature model results in an extended feature model that is useful during all the phases of the development. During the requirement engineering, it helps to elicit and to capture the required quality attributes and their variability. During the design stage it allows to take into account the quality aspects, the capture of the reasoning of some decisions, helps to specify the relationship impact among the architectural features and the quality attributes; and helps to make a first evaluation whether the quality attributes are going to be met. During the implementation stage, it allows to take into account the quality aspects and to capture the impact relationships. Of course, it is very important in the evaluation phase at the architecture level or after the implementation. The quality attribute characterization and the relationship between features help to guide the evaluation. Starting from this information, it is possible to

apply effort reduction strategies such as the construction of a generic model and/or the most significant product selection in order to evaluate if the required range of levels is met. After the evaluation, the extended feature model can be broadened with the derivation features and use the evaluation results for quantifying the impacts. This way, the extended feature model can be used for facilitating a quality aware derivation.

Conclusions of the case studies

The conclusions about the validation of the work load reduction at the evaluation stage are very positive. In all the case studies, the evaluation effort has been reduced (in relation to evaluating all the products). This reduction has been more considerable depending on the case study (the number of dependencies,...) and the evaluated quality attribute (number of impacts,...). The best results have been obtained using a generic evaluation model that allowed evaluating all the products with a minimum effort. When a subset of products are evaluated, good results have also been obtained: reductions of 72,5 % and 98 % in comparison to evaluating all the products. The reduction percentage has a strong relation with the number and the degree of the feature interaction when these have an effect on the evaluated quality attributes.

A.2. Results

During the research for this thesis, part of the results has been published in international conferences and journals in order to validate the direction and results of this research. The publications related to the thesis are detailed as well as the aspects of the thesis that are shown in the publications.

- Leire Etxeberria, Goiuria Sagardui, "*Product-Line Architecture: New Issues for Evaluation*", In Obbink, H., Pohl, K. (ed): Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, Proceedings, LNCS 3714 Springer (2005), 174-185
  - An analysis of the architecture evaluation in software product lines is presented: new aspects to consider in relation to the architecture evaluation of single systems:
    - Classification of quality attributes in software product lines.
    - New evaluation moments and techniques.

  - An analysis and survey of product line architecture evaluation methods and metrics.

- Leire Etxeberria, Goiuria Sagardui, "*Architectural Evaluation Framework for Product Lines*", In First International Conference on the Quality of Software Architectures, QoSA 2005, Erfurt, Germany, Proceedings Net.ObjectDays (2005), 505-512
  - Position paper that describes an evaluation framework for product line architectures.
- Leire Etxeberria, Goiuria Sagardui, Lorea Belategi, "*Modelling Variation in Quality Attributes*", First International Workshop on Variability Modelling of Software-Intensive Systems (VaMos 2007), Proceedings, (Eds.) Klaus Pohl, Patrick Heymans, Kyo-Chul Kang, Andreas Metzger, Lero Technical Report 2007-01, 2007
  - Analysis of the quality attributes variability: definition, requirements for its correct specification, and so on.
  - Analysis and survey of the existing approaches for specifying the quality attribute variability.
- Leire Etxeberria, Goiuria Sagardui, Lorea Belategi, "*Quality aware Software Product Line Engineering*", Journal of the Brazilian Computer Society (JBCS), Special Issue on Software Reuse, no. 1; Vol. 14; Mar. 2008 - ISSN 0104-6500
  - Analysis and proposal of the aspects to consider in order to meet the quality requirements in software product lines.
  - Analysis and survey of the variability modelling, design, architecture evaluation and testing methods for product lines.
- Leire Etxeberria, Goiuria Sagardui, "*Evaluation of Quality Attribute Variability in Software Product Families*", 15th IEEE International Conference on The Engineering of Computer-Based Systems, ECBS 08, proceedings, pp 255-264, 2008
  - The model for specifying the quality attribute variability and the method for facilitating the quality evaluation in software product lines in a cost effective way are presented.
  - Summary of the method's application to the Arcade game maker product line.
- Leire Etxeberria, Goiuria Sagardui, "*Quality Assessment in Software Product Lines*", H. Mei (Ed.): ICSR, 10th International Conference on Software Reuse, LNCS 5030, pp. 178-181, 2008.
  - The process to apply the method for facilitating the quality evaluation in product lines taking into account the quality variability is

shown.

- The application of the method is illustrated with the software calculator product line.

- Leire Etxeberria, Goiuria Sagardui, "*Variability Driven Quality Evaluation in Software Product Lines*", accepted for publication in 12th International Software Product Line Conference (SPLC 2008).
  - The effort reduction strategies of the CaLiPro method for evaluating the quality attributes are presented.

A.3. Limitations

During the CaliPro method's application, an extended feature model is proposed as a central variability model. Other product line artefacts and models (requirements, design, code, etc.) have also variability. Therefore, this variability should be reflected not only on those models but also in the proposed extended feature model. So traceability links must be maintained between the CaFM model and the rest of artefacts and models with variability. The definition and management of those traceability links have not been considered inside the CaLiPro method.

If several quality attributes are going to be evaluated at the same time, the number of features that affect those attributes can be very high. Therefore, the product selection strategy will not be effective. In the same way, in very big systems, the application of the methods and the results analysis can be complex. A higher automation is required and an improvement of the developed tool to give an appropriate support to bigger product lines.

The method allows obtaining information to perform a trade-off analysis and detect conflict among quality attribute. However, tool support is not provided to facilitate this task.

In critical domains and for certain quality attributes, the evaluation of all the products can be necessary. This can be in order to meet certification standards or due to the nature of the properties to analyze. In that case, the formal methods application can be required to verify and validate the quality aspects. The CaLiPro method does not consider the use of formal methods although it can be complementary technique to help to detect the most critical products and analyze those products in the first place.

A.4. Future work

The CaLiPro method is being applied in an industrial case and first results are encouraging. However, more work is needed to obtain definitive conclusions.

As future work, on the short-term the support tools that have been developed will be improved. At the moment, the fmp-CaLiPro tool provides a basic support for the process, but it can be improved adding graphical aspects, providing a better integration of the functionalities and adding new functionalities.

One of the foreseen functionalities is the conflict analysis between the quality attributes. Starting from the model and the defined impacts, an analysis of the quality attribute conflicts can be performed and present those conflicts in an explicit way.

Another important aspect where tools are required is the derivation stage. The guiCa tool helps during this phase, but we are also working to integrate the fmp-CaLiPro with the FAMA reasoning framework which allows the utilization of automatic reasoning techniques based on CSP (Constraint Satisfaction Problems). This way it is possible to perform a constraint-based product derivation and prevent to add features that make not to meet the required quality attributes.

The embedded system domain is one of the most favourable application domains for the method, because it is quite common to have product families with products that require different levels of quality attributes. Embedded systems have a hardware part and this hardware is also variable (different microprocessors) and can have an effect on the quality attributes. In our opinion, the hardware variability can be modelled in the same way as the functional variability with impacts on the quality attributes. However, an analysis to discover whether all the hardware variability requirements are covered is required. As part of the future work, a case study of an embedded system with the hardware variability will be performed.

The CaLiPro method is oriented mainly to evaluate product lines in development o already implemented ones but with not generated products (at least not all). However, el method can be also used during evolution and use the evaluation results of generated products to feedback the information base of the method (the CaFM model and especially the impacts) to use afterwards. So, to define a process and support is necessary.

In the medium term, based on the extended feature model, a process and sup-

port can be developed to help during the product line design where the quality aspects are included. As it has been concluded, the extended feature model is also useful for the requirement and design analysis. Providing some guidelines and tools, the decision making during the design stage can be improved, as well as the capturing of the reasoning behind those decisions.

On the long-term, and oriented to the critical embedded systems where safety and reliability are key aspects, the method and the model should be integrated with other models that allow the specification of the quality attribute variability. There are some UML profiles oriented towards the specification of the model quality attributes (QoS [Aagedal 02], MARTE [Faugere 07],...). However, those profiles do not support the variability that attributes can have. New models and model based validation methods are required in order to describe models with quality attributes and variability and validate the obtention of the quality attribute at an early stage.

Anexo B

Comparativa de enfoques de modelado de características

A continuación, se realiza una recopilación de enfoques basados en características; puesto que es uno de los modelos más utilizados y casi un estándar de facto para modelar la variabilidad en las líneas de productos software.

El modelado de características fue propuesto como parte del método FODA (Feature Oriented Domain Analysis) [Kang 90]. Tomando como base este método, varias extensiones y variantes han sido propuestas. En esta sección, FODA y sus extensiones se analizan en orden cronológico. El propósito de esta sección es destacar las variaciones que las extensiones han propuesto. Más información y comparativas de estas extensiones se pueden encontrar en [Schobbens 06] y [Czarnecki 04].

B.1. FODA

FODA define una característica como "*a prominent or distinctive and user-visible aspect, quality, or characteristic of a software system or systems*" [Kang 90]. Las características se muestran en un diagrama de características que es una jerarquía gráfica de *And/Or* de características. Las características pueden ser obligatorias, opcionales o alternativas. Las características opcionales se designan gráficamente con un pequeño círculo encima del nombre de la característica, como *Air Conditioning* (en la figura B.1). Las características alternativas se muestran como hijas de la misma característica padre, con un

arco dibujado a lo largo de todas las opciones, como en el caso de *Transmission*. El arco significa que solamente una de las características debe ser elegida (equivale a la operación *xor* lógica). El resto de características sin notación especial son todas obligatorias. Las reglas de composición se utilizan para definir la semántica existente entre características y que no se expresan en el diagrama: las relaciones de dependencia mutua (*Requires*) y exclusión mutua (*Mutex-with*). La información acerca del registro de *trade-offs*, razonamientos, justificaciones, etc. es también parte del modelo de características.

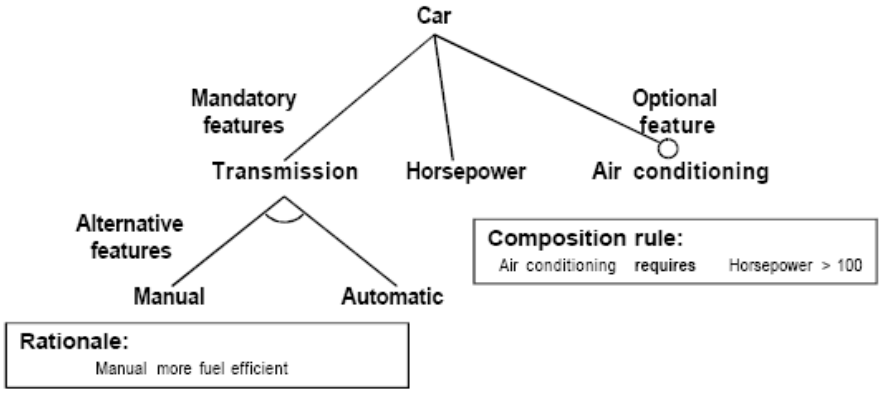


Figura B.1: Ejemplo del enfoque FODA

En FODA, las características se distinguen entre características de *contexto*, *representación* y *operacionales*. Las características de *contexto* describen la misión global o patrones de uso de la aplicación y aspectos como requisitos de rendimiento, exactitud y sincronización de tiempos que pueden afectar a las operaciones. Las características *operacionales* describen las funciones activas que se llevan a cabo (qué hace la aplicación). Y las características de *representación* describen cómo la información es visualizada por el usuario o producida por otra aplicación.

B.2. FORM

FORM (*Feature Oriented Reuse Method*) [Kang 98] fue la primera extensión de FODA realizada por sus mismos autores. Fue propuesta para aumentar el alcance del modelado de características e incluir el desarrollo de la arquitectura referencia. Su argumento era que el modelado de características no es sólo relevante para la ingeniería de requisitos, también para el diseño software. Además en FORM, redefinieron la definición original de característica para

incluir a otros participantes o *stakeholders* y no sólo usuarios: una característica es "*any prominent and distinctive concepts or characteristic that is visible to various stakeholders*" [Lee 02].

Las variaciones más importantes propuestas por este enfoque son los siguientes:

- Las relaciones de descomposición: *generalization/specialization* e *implemented-by* además de la relación *composed-of* (equivalente al propuesto por FODA).
- Cuatro categorías o capas de características: La capa de *capacidad*, la capa de *entorno de operación*, capa de *tecnología de dominio* y capa de *técnicas de implementación*. Las características de *capacidad* expresan servicio, operación y características no funcionales. Las características de *entorno de operación* expresan aspectos de hardware y software, las características de *tecnologías de dominio* expresan métodos de dominio, estándares, aspectos de leyes y las características de *técnicas de implementación* expresan decisiones de diseño e implementación.

Los autores de FODA también han extendido FODA para líneas de producto software: FOPLE (*Feature Oriented Product Line Software Engineering*) [Kang 02] [Lee 02]. Sin embargo, los cambios introducidos son muy similares a las variaciones de FORM.

B.3. FeatuRSEB

El método FeatuRSEB [Griss 98] es una integración de FODA en el método RSEB (*Reuse-Driven Software Engineering Business*). En este enfoque, el modelo de características toma un rol central y unificante durante toda la ingeniería de dominio.

Las variaciones más importantes introducidas por FeatuRSEB son:

- La introducción de la descomposición *or* además de la *xor*. Esto significa que es posible representar grupos de características donde más de una característica puede ser seleccionada (y no solamente una como en el caso *xor*)
- Nuevas categorías de características: *funcionales*, *arquitecturales* y de *implementación*. Las características *funcionales* expresan comportamiento, las *arquitecturales* expresan la estructura y configuración del sistema y

las características de *implementación* expresan decisiones de diseño e implementación.

- FeatuRSEB también propone cambios en la representación gráfica de las características y propone una representación gráfica para las relaciones de restricciones o reglas de composición (*requires* y *mutex*).

B.4. *Generative Programming*

El enfoque *Generative Programming* [Czarnecki 00] ha estudiado los modelos de características en el contexto de la programación generativa, un paradigma de programación que tiene como objetivo automatizar el desarrollo software de las líneas de producto. Define una característica o *feature* como "*a distinguishable characteristic of a concept (e.g., system, component and so on) that is relevant to some stakeholder of the concept*".

Algunas de las variaciones propuestas en este enfoque son la introducción de la descomposición *or* además de la *xor* (como en FeatuRSEB). También proponen el uso de parámetros y reglas de dependencia por defecto que proponen valores por defecto para los parámetros no especificados, basándose en otros parámetros. Otra variación es el uso de restricciones horizontales y verticales. Las restricciones horizontales describen dependencias entre características de un nivel de abstracción similar mientras que las restricciones verticales y las reglas de dependencia por defecto mapean características de especificación de alto nivel con las de implementación.

Los autores de este enfoque también han propuesto nuevas variaciones en el enfoque llamado CBFM (*Cardinality based Feature Modelling*) [Czarnecki 04] que se analiza posteriormente.

B.5. Enfoque de Van Gurp

El enfoque de [Gurp 01] se basa en FeatuRSEB. Las principales diferencias son:

- Introducción de características externas: características de la platafor-

ma objetivo, que no son parte directamente del sistema pero que son importantes porque el sistema las utiliza y depende de ellas.

- Añade *binding times*.
- Diferencias gráficas para representar *xor* y *or*.

B.6. Enfoque de Riebisch

Riebisch et al [Riebisch 02] introducen las multiplicidades o cardinalidad de grupo para evitar los problemas de ambigüedad detectados en las combinaciones de características obligatorias y opcionales con alternativas, *or* y *xor*. Remplazan las relaciones *or*, *xor* y *and* por multiplicidades parecidas a las de UML o cardinalidades de grupo y aristas obligatorias y opcionales.

- *Multiplicidades:* Proponen una forma de anotar las multiplicidades de un conjunto de características para evitar ambigüedades y permitir otras multiplicidades además de las cuatro más comunes (obligatoria, opcional, *xor* y *or*). El conjunto de características es anotado especificando el mínimo y máximo número de características que puede seleccionarse del conjunto.
- *Aristas opcionales y obligatorias (Optional and mandatory edges):* La propiedad de obligatorio, opcional o alternativa se almacena como una propiedad de las relaciones de características y no como una propiedad de la característica. De este modo es posible que sea opcional en un lado y obligatorio en otro.

Riebisch [Riebisch 03] redefine qué es una característica: "*a feature represents an aspect valuable to the customer*", y distingue cuatro categorías de características: *Funcional, interface, parámetro y concepto.* Las características *funcionales* expresan el comportamiento, las características de *inferface* expresan la conformidad del producto a un estándar o un subsistema, las características de *parámetro* expresan propiedades enumerables o no funcionales, y las características de *contexto* son utilizados para encapsular características abstractas de la jerarquía como por ejemplo la raíz.

B.7. CBFM

El enfoque CBFM o *Cardinality based Feature Modelling* [Czarnecki 04] se basa en el enfoque Generative programming [Czarnecki 00] y en otros trabajos de sus autores y colaboradores como [Czarnecki 02] y [Bednasch 02].

En este enfoque se define característica como "*a system property that is relevant to some stakeholder and is used to capture commonalities and discriminate between systems*". Las principales variaciones introducidas son:

- *Cardinalidades de las características:* Las características se anotan con cardinalidades. Esta cardinalidad indica cuántas copias de una característica solitaria pueden hacerse [Czarnecki 02]. Las características obligatoria y opcional son características especiales con la cardinalidad [1..1] y [0..1].
- *Cardinalidades de grupo:* Conjuntos de características agrupadas que son anotadas con cardinalidades (como propone [Riebisch 02]).
- *Atributos:* Este concepto fue introducido por [Czarnecki 02] para representar un valor de un dominio grande o infinito como integers o strings.
- *Referencias de diagrama de características:* Este concepto fue adaptado a partir de la modularización introducida por [Bednasch 02] que permite romper diagramas grandes y reutilizar partes comunes en varios sitios.

Realizan una clasificación de características relacionadas con la forma en la que las características se implementan: características *concretas* como almacenamiento de datos o ordenamiento se realizan como componentes individuales. Las características *aspectuales* como la sincronización o el *logging* pueden afectar a varios componentes y pueden modularizarse usando tecnologías de aspecto. Las características *abstractas* como requisitos de rendimiento se mapean sobre una configuración de componentes y/o aspectos y las características de *agrupamiento* pueden representar puntos de variación y mapearse a una interfaz común de componentes compatibles o pueden tener un propósito puramente organizacional.

B.8. PLUSS

El enfoque PLUSS (*Product Line Use case modelling for Systems and Software engineering*) [Eriksson 05] se basa en FeatuRSEB. En PLUSS el propósito primordial del modelo de características es ser una herramienta de visualización de variantes en el modelo abstracto de casos de uso de la línea. La principal variación añadida por PLUSS es:

- Nodos operando: Adaptadores únicos que representan descomposición *xor* de sus padres y adaptadores múltiples que representan descomposición *or*.

B.9. Los atributos de calidad en los enfoques

Desde el punto de vista de los atributos de calidad, las definiciones de característica han sido analizadas para ver si los atributos de calidad encajan en estas definiciones. En la definición de FODA ("*a prominent or distinctive and user-visible aspect,* ***quality****, or characteristic of a software system or systems*") un atributo de calidad es considerado una característica y en [Czarnecki 05b] también: "*a feature may denote any* ***non-functional*** *characteristic*". Por lo tanto, los atributos de calidad pueden encajar en la definición de las características.

Las categorías de características también han sido analizadas para ver si los atributos de calidad se mencionan. En concreto se mencionan explícitamente como parte de las siguientes categorías: características *contexto en* FODA, características de *capacidad* en FORM, características *parámetro* en Riebisch [Riebisch 03] y características *abstractas* en Cardinality-based Feature Modelling [Czarnecki 04].

B.10. Resumen

En la tabla B.1 se muestra un resumen de los enfoques y sus características.

Tabla B.1: Enfoques basados en modelos de características

Enfoque	**Basado en**	**Definición (Vista usuario y/o desarrollador)**	**Categorías de características**	**Nodos**	**Relaciones de descomposición**	**Otras cosas que añaden**
FODA	-	Vista usuario	Contexto,Representación y Operacional	Opcional, Obligatorio y Alternativo	Consists-of (And, Xor)	
FORM	FODA	Vista usuario y vista desarrollador	Capacidades, Entorno Operativo, Tecnología de dominio y Técnicas de Implementación	Opcional, Obligatorio y Alternativo	Composed-of (and, xor), Generalization/Specialization e Implemented-by	
FeatuRSEB	FODA	Vista usuario y vista desarrollador	Funcional, Arquitectural e Implementación	Opcional, Obligatorio y Variante Las características con hijos y descompuestas a través de un or o xor se llaman puntos de variación y los hijos se llaman variantes	Composed-of (And, Xor, Or)	Representación gráfica
Generative programming	FODA	Vista usuario y vista desarrollador		Opcional, Obligatorio y alternativo	And, Xor y Or	Parámetros, reglas de dependencia por defecto
Enfoque de Van Gurp et al	FeatuRSEB	Vista usuario y vista desarrollador		Opcional, Obligatorio, Variante	Composed-of (And, Xor, Or)	Características externas, Binding times y diferencias gráficas
Riebish	Generative programming	Vista usuario y vista desarrollador	Funcional, Interface, Parámetro, Concepto	Opcional, Obligatorio y alternativo Se almacena como propiedad de la relación de una característica (una característica puede ser opcional por un lado y obligatoria por otro)	Multiciplidades o cardinalidades de grupo	
CBFM	Generative programming	Vista usuario y vista desarrollador	Concreto, Aspectual, Abstracto y de agrupamiento (grouping)	Cardinalidades de característica y Atributos	Cardinalidades de grupo	Referencias de diagrama de características

PLUSS	FeatuRSEB	Vista usuario y vista desarrollador		Opcional, Obligatorio y Variante	Nodos operando: Adaptadores únicos que representan xor-decomposition de sus padres y adaptadores múltiples que representan or-decomposition.	

Anexo C

Descripción SPEM del proceso CaLiPro

C.1. Introducción

El objetivo de este anexo es explicar la metodología CaLiPro (Método de evaluación de atributos de **Ca**lidad en **Lí**neas de **Pro**ductos software). Este método sirve para modelar y gestionar la variabilidad (funcional y de calidad) durante todo el ciclo de vida, facilitando la validación de la calidad de la línea y permitiendo una derivación que tiene en cuenta los niveles de calidad de los atributos de calidad operacionales.

La figura C.1 muestra como se incluye el proceso propuesto en el ciclo de vida de la ingeniería de línea de productos software de [Czarnecki 98]. El proceso es transversal a actividades como el análisis, el diseño y la implementación por lo que se puede incluir en la mayoría de los ciclos de vida de ingeniería de dominio.

A continuación se explican las tres fases principales del proceso: especificar, validar y derivar:

- **Fase Especificación de la Variabilidad:** Esta fase construye un modelo de variabilidad (Modelo de características extendido: CaFM) que recoge la variabilidad a diferentes niveles de abstracción: especificación de requisitos tanto funcionales como de calidad, decisiones de diseño y de implementación variantes y características de evaluación. El mode-

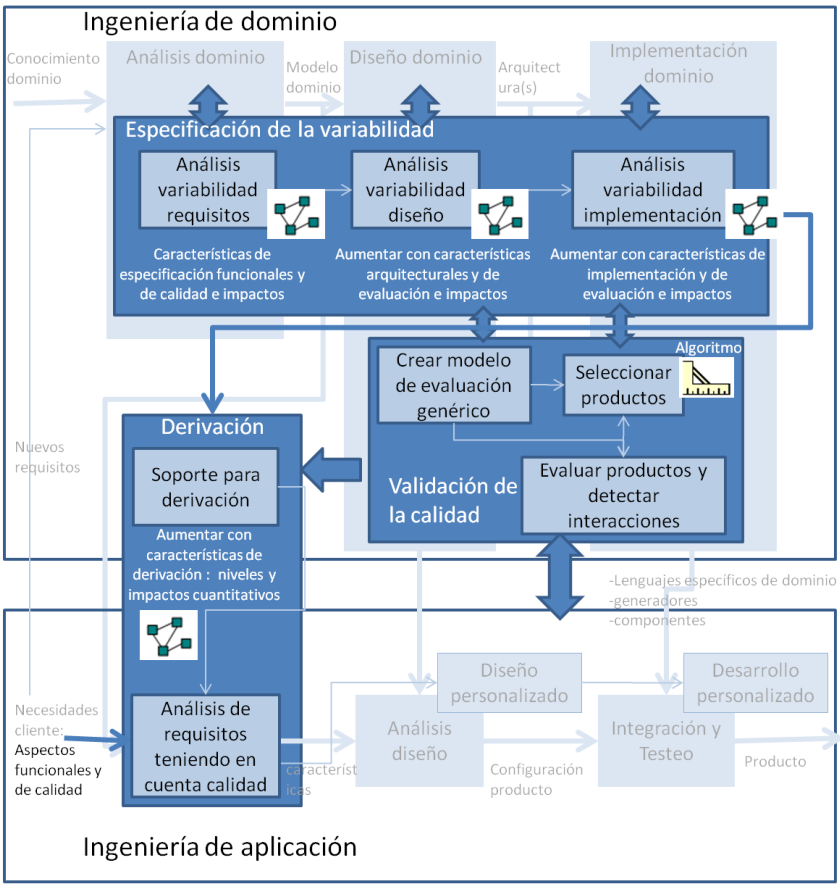


Figura C.1: Fases del proceso CaLiPro

lo permite relacionar las características funcionales con los atributos de calidad.

- **Fase Validación de la Calidad:** Esta fase testea y comprueba que la línea soporte los aspectos de calidad clave de los productos. Para que la validación se pueda hacer de manera rentable, se define un modelo de evaluación genérico (con variabilidad, válido para evaluar cualquier producto de la línea) y/o se seleccionan productos clave que permitan extrapolar los resultados de la evaluación a la línea. Esta tarea se realiza mediante un algoritmo que selecciona el mínimo número de productos para detectar interacciones entre las características y cuantificar los impactos para poder extrapolar los datos a todos los productos.
- **Fase Derivación:** Primeramente se aumenta el modelo de características extendido con características de derivación de los atributos de calidad. Posteriormente, los productos se derivan teniendo en cuenta la influencia de las características funcionales en la calidad final del producto.

C.2. El proceso CaLiPro

Para describir el proceso CaLiPro (ver figura C.2) se ha utilizado SPEM (Software Process Engineering Metamodel),un estándar de la OMG (Object Management Group) [OMG 05], ya que permite expresar procesos de desarrollo software de una forma estándar.

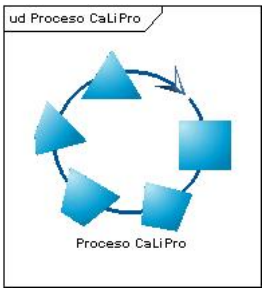


Figura C.2: Proceso CaLiPro

Como se ha mencionado con anterioridad, el proceso CaLiPro consta de tres fases principales (figura C.3): la especificación de la variabilidad, la validación de la calidad y la derivación. En la figura C.3 se pueden ver estas fases así como las entradas y salidas para las mismas. Uno de los productos de trabajo

clave es el Modelo CaFM, con el que se trabaja en todas las fases. Este modelo se explica con más detalle en la sección C.3.

El proceso CaLiPro se puede llevar a cabo en paralelo con el desarrollo de una línea de productos o a partir de una ya existente. Por lo tanto, hay una interacción con los activos de la línea que se están generando o ya existen. Estos activos van a ser una entrada en varias de las fases. En la sección C.4 se profundiza en el detalle de estos activos.

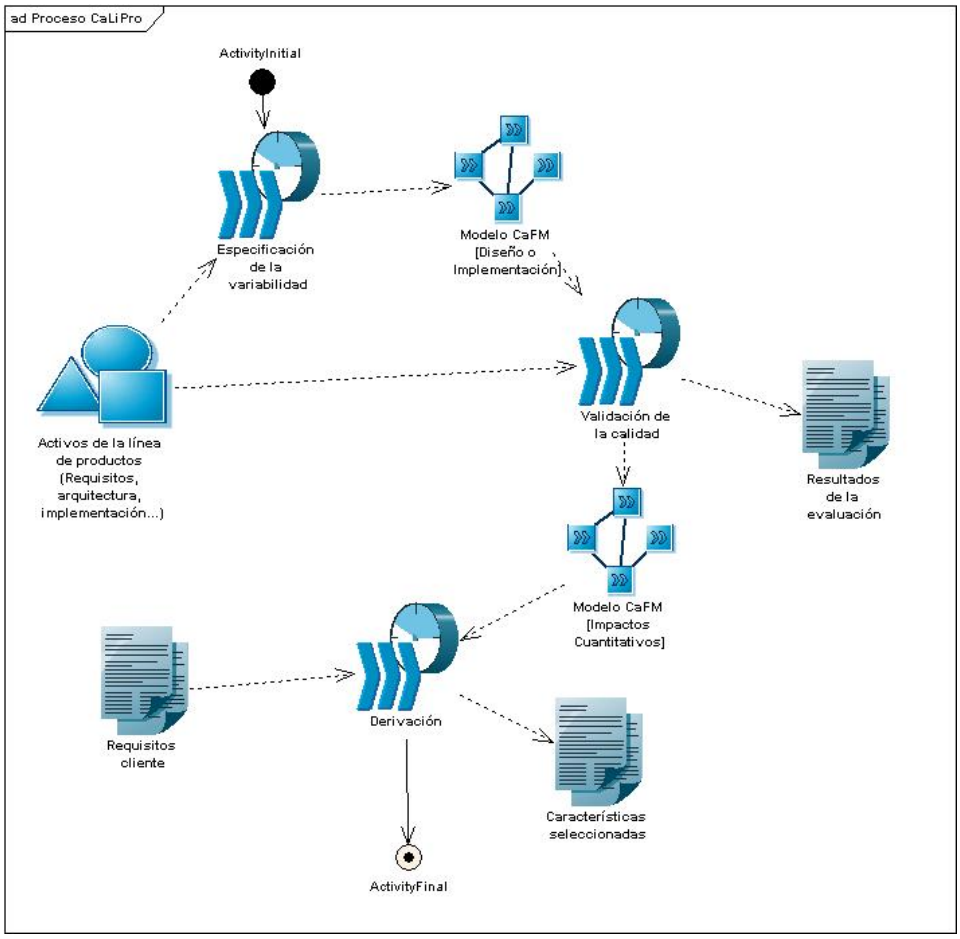


Figura C.3: Diagrama de actividad que describe las fases y productos de trabajo del proceso CaLiPro

C.3. El modelo CaFM

El modelo CaFM (CaLiPro Feature Model) es un modelo de características extendido que permite especificar de forma más precisa la variabilidad en los atributos de calidad y facilitar de este modo una evaluación de los atributos de calidad de la línea y una derivación teniendo en cuenta aspectos de calidad.

Este modelo se crea en la primera actividad del proceso CaLiPro y es utilizado y aumentado a lo largo de todo el proceso. Dependiendo del momento en el que se usa y los aspectos que se le añaden se pueden diferenciar cinco modelos CaFM (ver figura C.4):

- Modelo CaFM [Requisitos]: Este modelo de características solamente tiene características funcionales y de especificación de calidad (a nivel de requisitos), así como impactos entre estas características.
- Modelo CaFM [Diseño]: Modelo CaFM [Requisitos] aumentado. Este modelo tiene además de características de nivel de requisitos, características arquitecturales, de caracterización de atributos de calidad e impactos entre ellos.
- Modelo CaFM [Implementación]: Modelo CaFM [Diseño] aumentado con características de implementación, algunas caracterizaciones de calidad más e impactos entre ellos.
- Modelo CaFM [Evaluación]: Modelo CaFM [Diseño] o Modelo CaFM [Evaluación] en el que se han seleccionado los aspectos de calidad a evaluar y el alcance de la evaluación. Dentro de este modelo, una vez que se ha realizado la evaluación se pueden cuantificar los impactos:
  - Modelo CaFM [Impactos cuantitativos]: Modelo CaFM [Evaluación] en el que se recogen los impactos cuantitativos (en los modelos anteriores los impactos son cualitativos).
- Modelo CaFM [Derivación]: Modelo CaFM [Evaluación] aumentado con características de derivación.

C.4. Activos de la línea de productos

Como ya se ha mencionado, el proceso CaLiPro se lleva a cabo en paralelo con el desarrollo de una línea de productos o a partir de una línea de productos

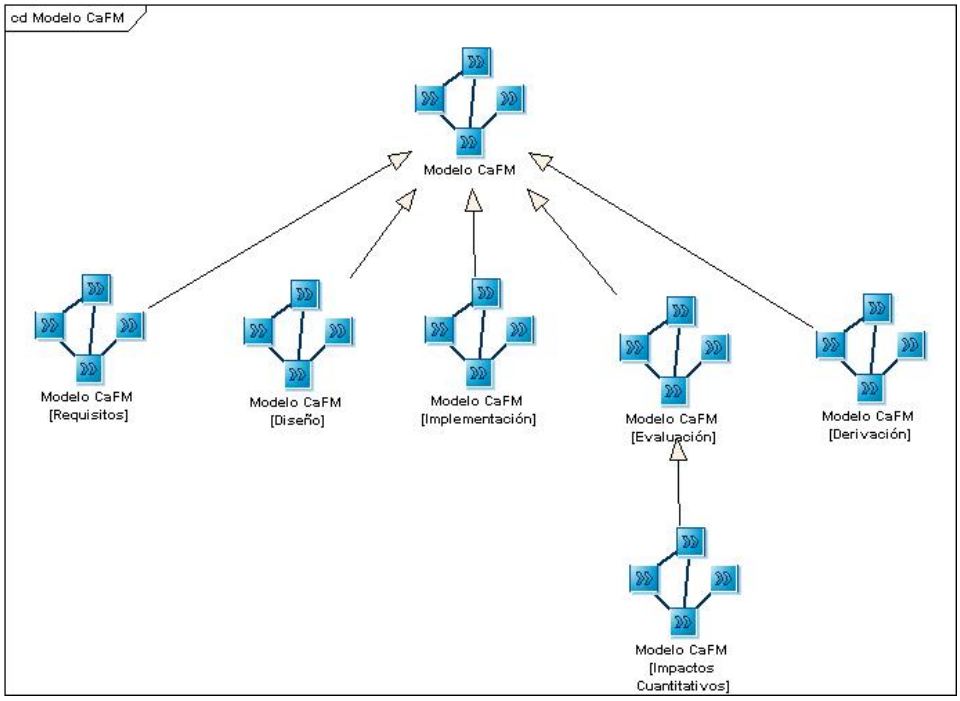


Figura C.4: Diagrama de clases del modelo CaFM

ya existente. Por lo tanto, los activos de la línea que se están generando o ya existen serán una entrada al proceso. En la figura C.5 se detallan los activos que se mencionan explícitamente en la definición del proceso CaLiPro utilizando SPEM.

C.5. Fases del proceso CaLiPro

C.5.1. Fase de Especificación de la variabilidad

Los objetivos de esta fase son los siguientes:

1. Definir la variabilidad funcional, arquitectural y de implementación mediante características funcionales, arquitecturales y de implementación y organizar las características en el modelo de características extendido.
2. Definir y caracterizar la variabilidad de calidad de la línea mediante características de calidad para permitir una validación posterior y organizar

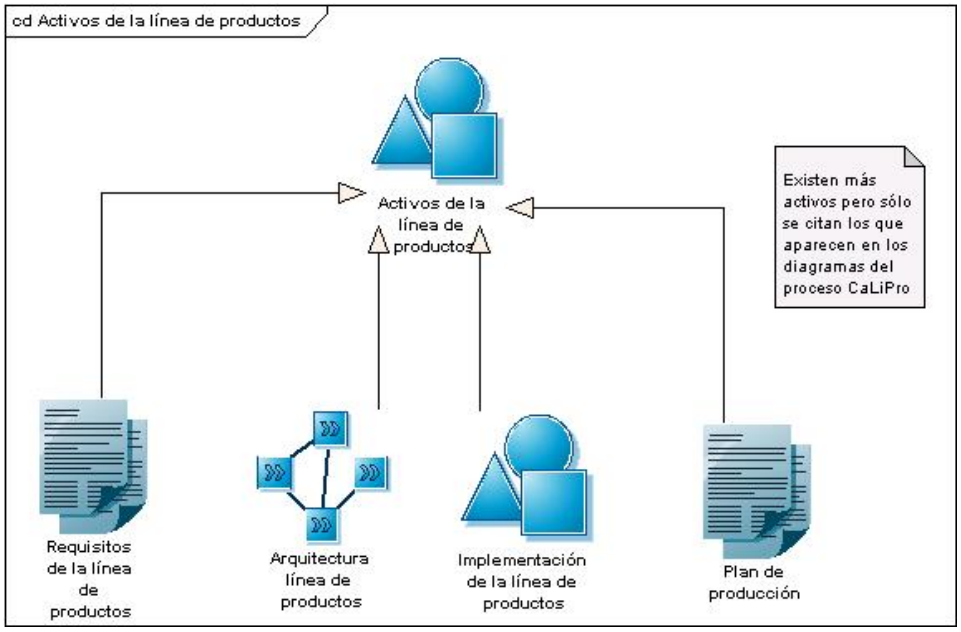


Figura C.5: Diagrama de clases de los activos de la línea de productos

las características en el modelo de características extendido.

3. Definir la variación indirecta, es decir, los impactos desde las características funcionales, arquitecturales y de implementación hacia las características de calidad.

Esta fase tiene tres definiciones de trabajo (ver figura C.6) relacionados con la etapa del desarrollo cuya variabilidad hay que especificar.

Se parte de los activos de la línea de productos para analizar y especificar la variabilidad mediante el modelo CaFM. Si la evaluación de la calidad se va a realizar a nivel de diseño o todavía no existe una implementación, la fase se finaliza antes sin llegar a un Modelo CaFM de implementación (Ver figura C.7).

A continuación se definen las actividades y los roles que participan en las tres definiciones de trabajo de esta fase (ver figuras: C.8,C.9 y C.10).

En la tabla C.1 se muestra un resumen de las actividades de esta fase.

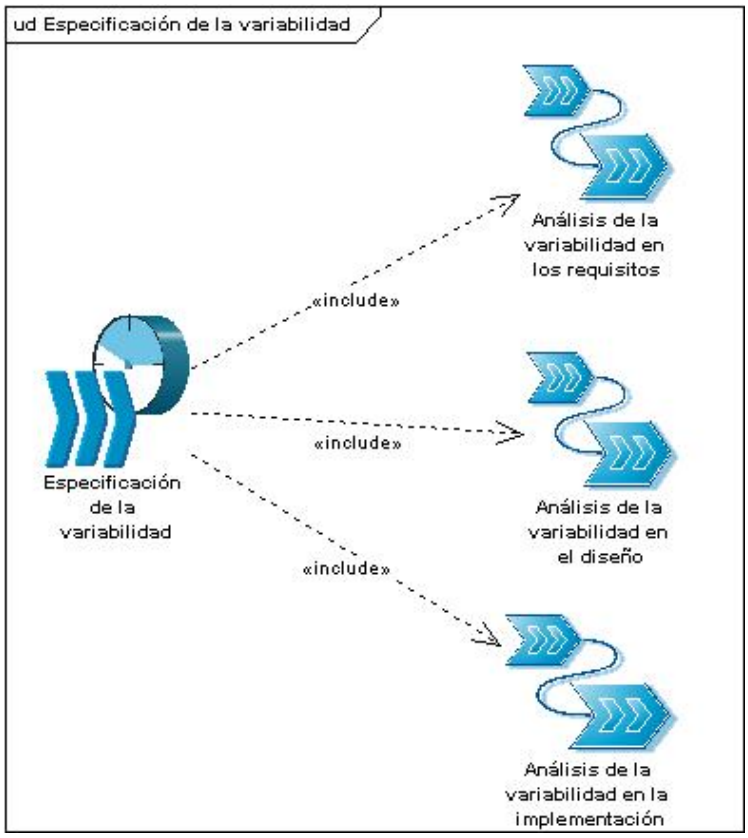


Figura C.6: Diagrama de casos de uso con las definiciones de trabajo que incluye la fase de especificación de la variabilidad

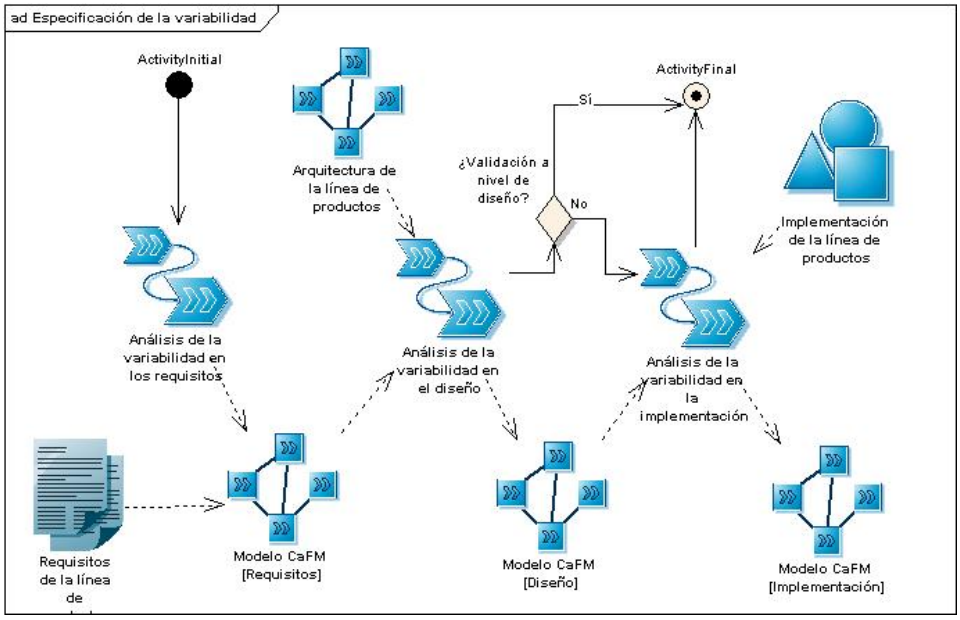


Figura C.7: Diagrama de actividad que describe la fase de especificación de la variabilidad en términos de definiciones de trabajo y productos de trabajo

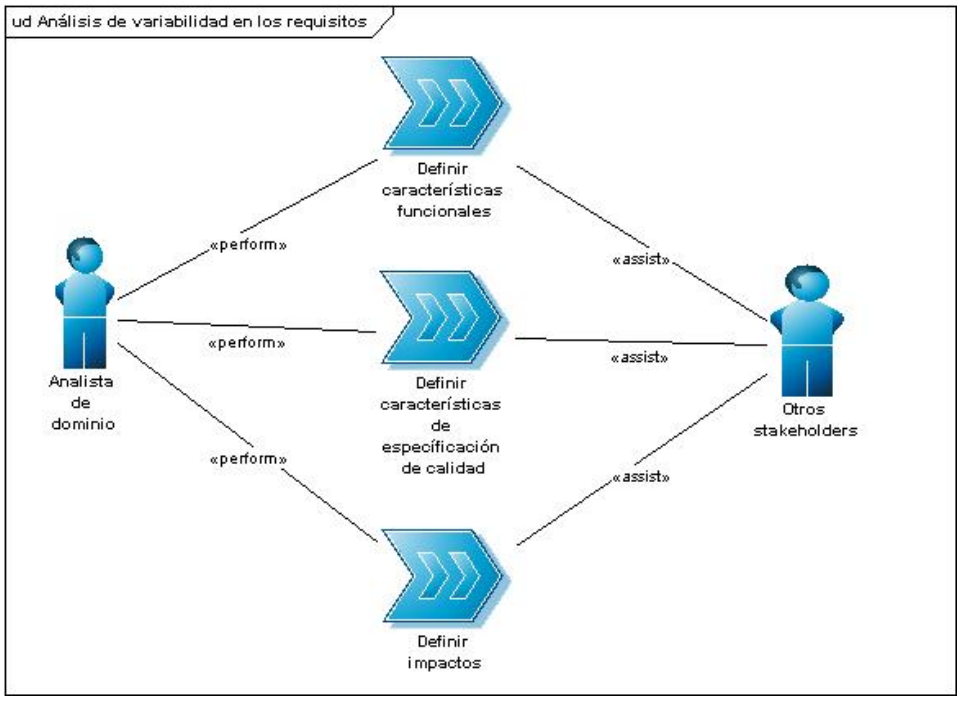


Figura C.8: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Análisis de la variabilidad en los requisitos"

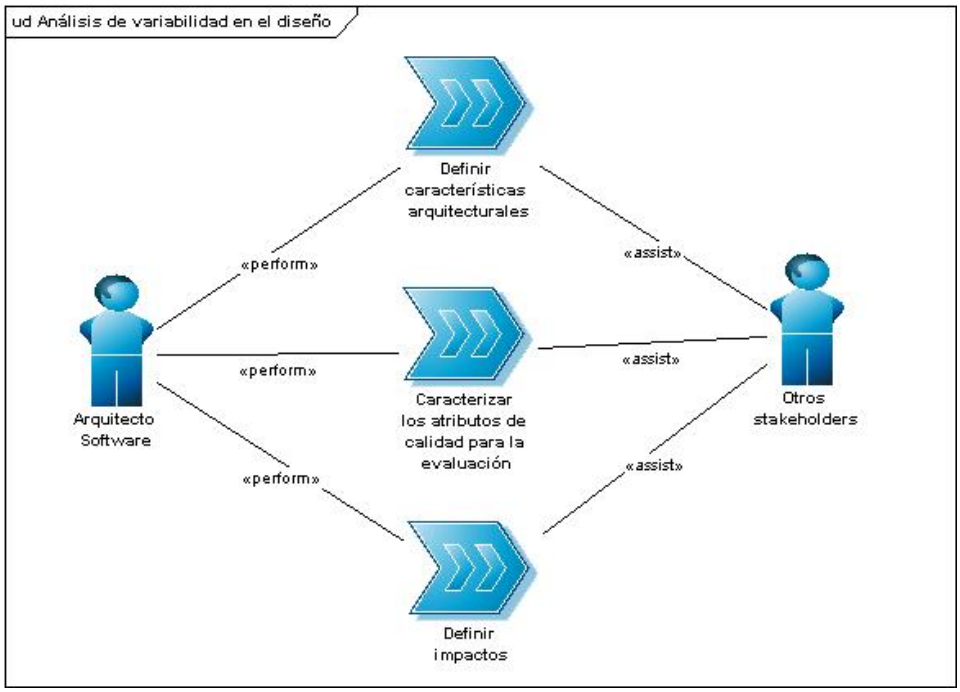


Figura C.9: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Análisis de la variabilidad en el diseño"

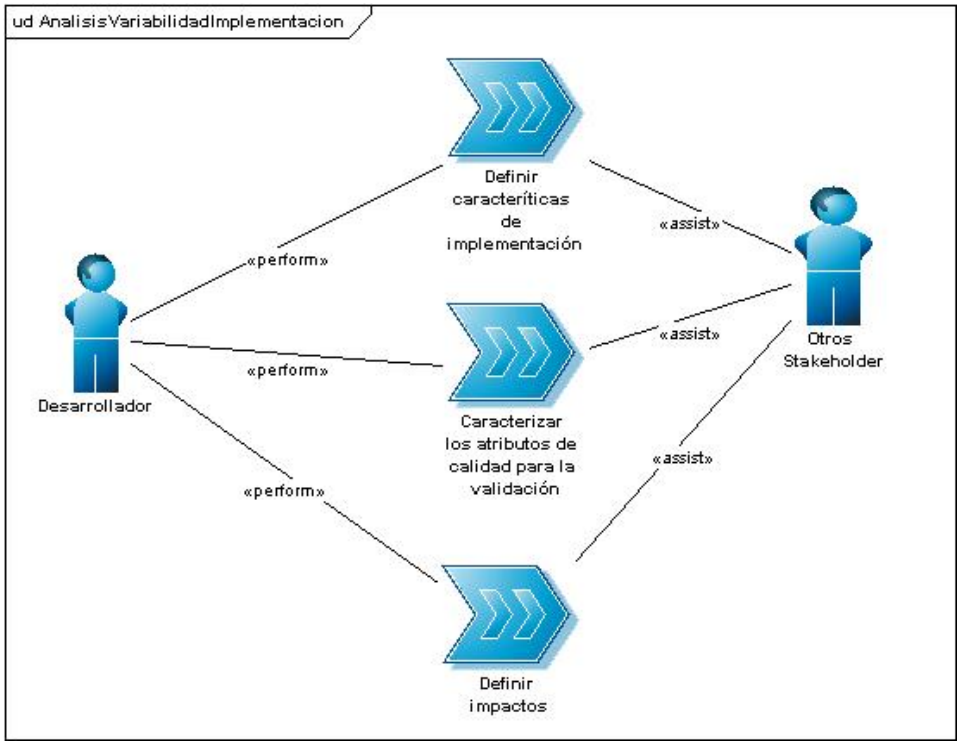


Figura C.10: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Análisis de la variabilidad en la implementación"

Tabla C.1: Resumen de las actividades de la fase de Especificación de la variabilidad

Definición de trabajo	**Actividad**	**Descripción de la actividad**	**Rol**
Análisis de variabilidad en los requisitos	Definir características funcionales	Se identifican las características funcionales	Analista de dominio
	Definir características de especificación de calidad	Se identifican las características de calidad	Analista de dominio
	Definir impactos	Se definen los primeros impactos	Analista de dominio
Análisis de variabilidad en el diseño	Definir características arquitecturales	Se identifican las características a nivel de diseño	Arquitecto software
	Caracterizar los atributos de calidad para la evaluación	Se describen las características de calidad para su posterior evaluación	Arquitecto software
	Definir impactos	Se identifican los impactos a nivel de diseño	Arquitecto software
Análisis de variabilidad en la implementación	Definir características de implentación	Se identifican las características a nivel de implementación	Desarrollador
	Caracterizar los atributos de calidad para la validación	Se describen las características de calidad para su posterior validación	Desarrollador
	Definir impactos	Se identifican los impactos a nivel de implementación	Desarrollador

Roles involucrados

Tres roles son los encargados de llevar a cabo las actividades: Analista del dominio, arquitecto software y desarrollador. Además de estos roles, otros *stakeholders* también participan asistiendo en las actividades. A continuación se detallan las responsabilidades de estos roles.

- Analista del dominio: Es el responsable de analizar el dominio e identificar la parte común y variable de los productos de la línea. En el proceso CaLiPro es el encargado de:
  1. Crear y refinar el modelo CaFM para especificar la parte común y variable de la línea a nivel de requisitos tanto desde un punto de vista funcional como de calidad.
- Arquitecto software: Es el responsable de diseñar la arquitectura de línea de productos. En el proceso CaLiPro es el encargado de:
  1. Plasmar la variabilidad arquitectural en el modelo CaFM.
  2. Caracterizar los atributos de calidad para su posterior evaluación.
  3. Determinar el impacto de la variabilidad arquitectural en los aspectos de calidad.
- Desarrollador: Es el responsable de codificar la línea de productos implementando los mecanismos de variabilidad necesarios. En el proceso CaLiPro es el encargado de:
  1. Plasmar la variabilidad de implementación en el modelo CaFM.
  2. Caracterizar los atributos de calidad para su posterior validación.
  3. Determinar el impacto de la variabilidad de implementación en los aspectos de calidad.

Además de estos tres roles, otros stakeholders también pueden estar presentes y ayudar a lo largo de esta fase.

- Expertos de dominio
- Responsable de calidad
- Encargados de velar por atributos de calidad específicos: ingeniero de rendimiento, experto en seguridad, etc.
- Testeador
- Encargado de mantenimiento
- Gestores de la línea
- Encargados de marketing

- Clientes
- Usuarios finales
- Constructor de aplicaciones
- Expertos de dispositivos
- Etc.

Por supuesto, una persona puede tener varios roles. Además, no todos los stakeholders participan en todas las definiciones de trabajo o de igual forma en una definición de trabajo o en otra.

Los *stakeholders* que tienen una vista de usuario final como los clientes, encargados de marketing y usuarios pueden ayudar en la toma de decisiones de variabilidad. Los gestores de la línea al tener una visión global también pueden ayudar del mismo modo.

Respecto a la variabilidad en la calidad, los *stakeholders* más importantes y los que pueden aportar más son los expertos: expertos de dominio, expertos en un atributo de calidad (ingeniero de rendimiento, experto en seguridad, etc.), expertos de dispositivos, los responsables de mantenimiento (expertos en mantenibilidad), el testeador (experto en testeabilidad), etc. Estos expertos debido a su experiencia pueden dar información muy valiosa sobre impactos y ayudar a definirlos.

A continuación se detallan cada una de las definiciones de trabajo:

C.5.1.1. Análisis de la variabilidad en los requisitos

El objetivo de esta definición de trabajo es analizar y especificar la variabilidad a nivel de requisitos tanto funcionales como de calidad, así como los primeros impactos (variación indirecta) entre características funcionales y de calidad (ver figura C.11).

Entregables

La salida de esta definición de trabajo es un modelo CaFM con la variabilidad a nivel de requisitos.

Precondiciones

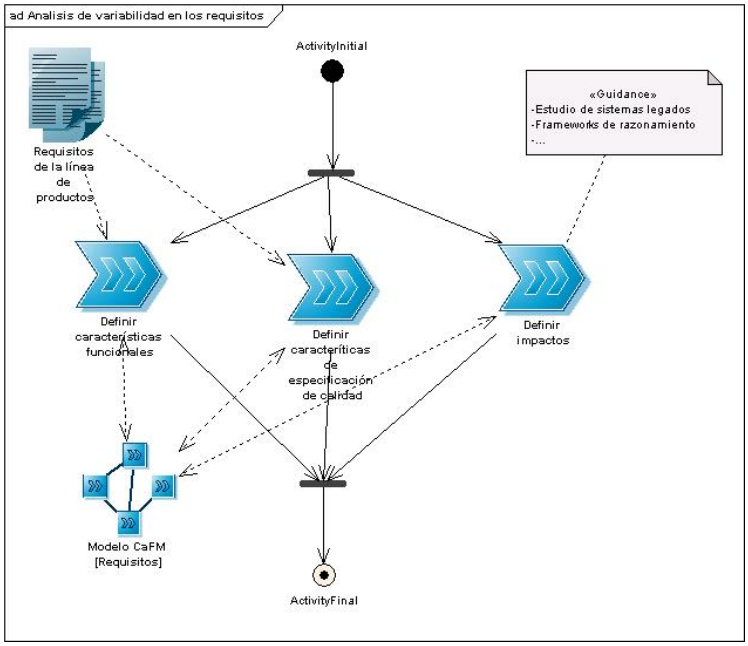


Figura C.11: Diagrama de actividad de la definición de trabajo "Análisis de la variabilidad en los requisitos"

Debe existir información acerca de los requisitos de la línea de productos recogida en un documento o modelo. Este documento puede ser un borrador ya que esta definición de trabajo se puede realizar en paralelo con el análisis del dominio durante el desarrollo de una línea de productos software.

Guías

El analista del dominio con ayuda de otros *stakeholders* debe definir la variabilidad funcional y de calidad utilizando un modelo CaFM, así como definir los impactos entre las características funcionales y de calidad.

A la hora de definir los impactos, es muy útil la opinión de los expertos, los estudios de sistemas legados previos y los marcos de razonamiento sobre atributos de calidad que se han definido para ayudar durante el diseño o evaluación como por ejemplo [Babar 05] para seguridad.

Entrada/Salida

La entrada es el documento o modelo con los requisitos de la línea de productos y la salida es un modelo CaFM con la variabilidad a nivel de requisitos descrita.

C.5.1.2. Análisis de la variabilidad en el diseño

El objetivo de esta definición de trabajo es muy similar a la anterior definición de trabajo pero a nivel de diseño. Consiste en analizar y especificar la variabilidad en el diseño. La variabilidad en las decisiones de diseño de un sistema puede afectar mucho a los atributos de calidad y a su variabilidad; por lo tanto, la definición de impactos cobra vital importancia (ver figura C.12).

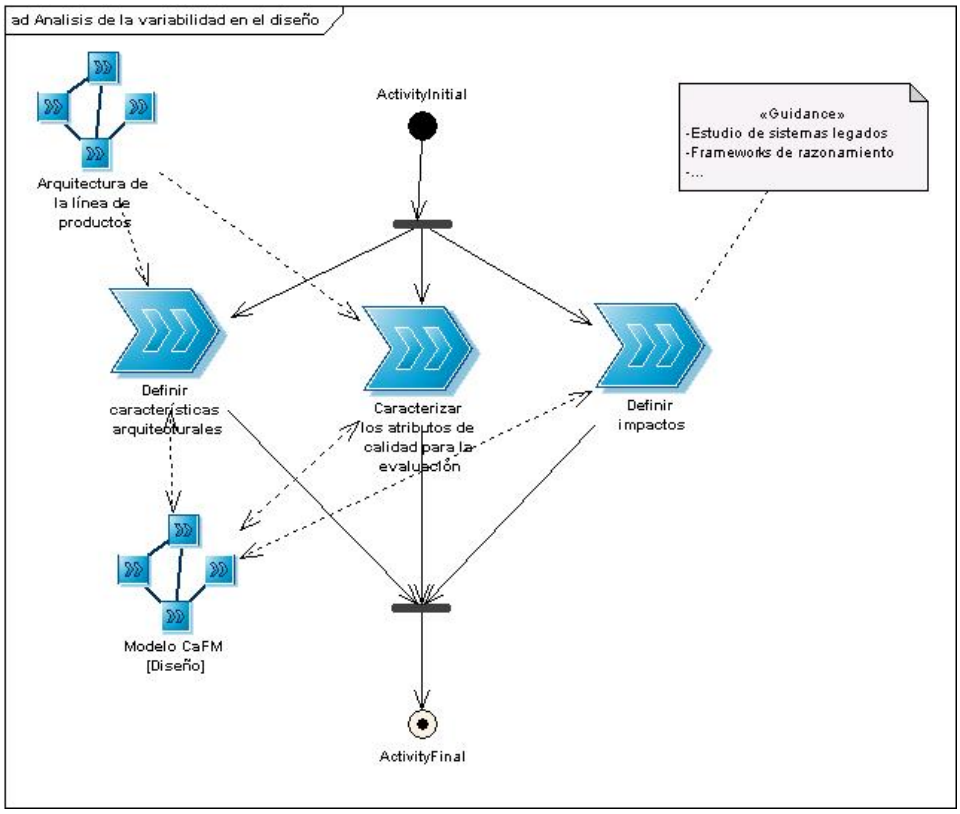


Figura C.12: Diagrama de actividad de la definición de trabajo "Análisis de la variabilidad en el diseño"

Entregables

La salida de esta definición de trabajo es un modelo CaFM con la variabilidad a nivel de diseño.

Precondiciones

Debe existir información acerca del diseño y de la arquitectura de línea de productos recogida en un documento o modelo. Este documento puede ser un

borrador ya que esta definición de trabajo se puede realizar en paralelo con el diseño de la línea de productos software.

Guías

El arquitecto software con ayuda de otros *stakeholders* debe definir la variabilidad en el diseño y caracterizar los atributos de calidad para su posterior evaluación; así como definir los impactos entre las características arquitecturales y de calidad.

A la hora de definir los impactos, es muy útil la opinión de los expertos, los estudios de sistemas legados previos y los marcos de razonamiento sobre atributos de calidad.

Dependencias

Esta definición de trabajo depende de la definición anterior ya que aumenta el modelo CaFM a nivel de requisitos (salida de definición anterior).

Entrada/Salida

Las entradas son la documentación sobre la arquitectura o diseño de la línea de productos y el modelo CaFM a nivel de requisitos (resultado de la definición de trabajo anterior). La salida es un modelo CaFM especificando la variabilidad a nivel de diseño además de a nivel de requisitos.

C.5.1.3. Análisis de la variabilidad en la implementación

El objetivo de esta definición de trabajo es muy similar a las definiciones de trabajo anteriores pero a nivel de implementación. Consiste en analizar y especificar la variabilidad en la implementación (ver figura C.13).

Entregables

La salida de esta definición de trabajo es un modelo CaFM con la variabilidad a nivel de implementación. Por lo tanto, es un modelo CaFM que recoge toda la variabilidad de la línea.

Precondiciones

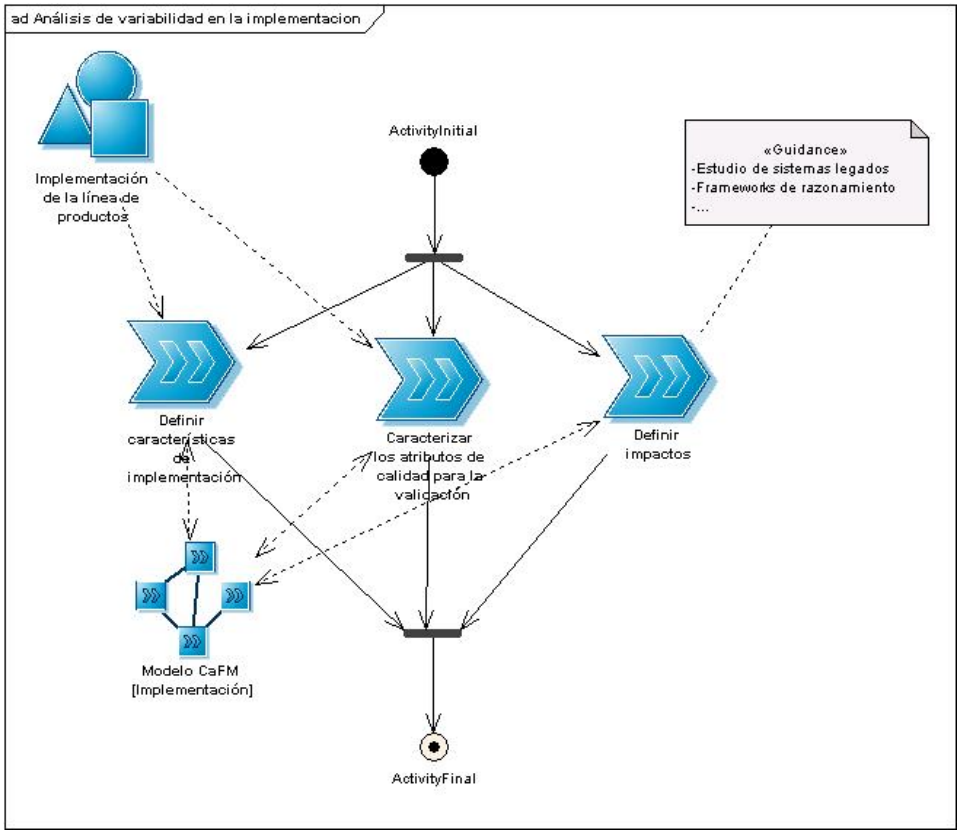


Figura C.13: Diagrama de actividad de la definición de trabajo "Análisis de la variabilidad en la implementación"

Debe existir información acerca de la implementación de la línea de productos, es decir, la línea de productos debe estar codificada. Esta definición de trabajo no es obligatoria para pasar a la siguiente fase, si no existe codificación se puede realizar una evaluación a nivel de diseño.

Guías

El desarrollador o codificador con ayuda de otros *stakeholders* debe definir la variabilidad en la implementación y caracterizar los atributos de calidad para su posterior evaluación; así como definir los impactos entre las características de implementación y de calidad.

A la hora de definir los impactos, en este caso también es útil la opinión de los expertos, los estudios de sistemas legados previos y los marcos de razonamiento sobre atributos de calidad.

Dependencias

Esta definición de trabajo depende de la definición anterior ya que aumenta el modelo CaFM a nivel de diseño (salida de definición anterior).

Entrada/Salida

Las entradas son la implementación de la línea de productos y el modelo CaFM a nivel de diseño (resultado de la definición de trabajo anterior). La salida es un modelo CaFM especificando la variabilidad a todos los niveles incluyendo también la implementación.

C.5.2. Fase de Validación de la calidad

Los objetivos de esta fase son los siguientes:

1. Determinar qué atributos de calidad se van a evaluar, con qué alcance, a qué nivel (diseño o implementación) y con qué método de evaluación.
2. Crear un modelo de evaluación genérico si la evaluación se realiza a nivel de diseño.
3. Seleccionar los productos o diseños a evaluar (si es necesario).
4. Realizar la evaluación y analizar los resultados.

Esta fase tiene cuatro definiciones de trabajo (ver figura C.14) para seleccionar los atributos a evaluar y el método, crear un modelo genérico, seleccionar los productos o diseños a evaluar y evaluar los productos o diseños.

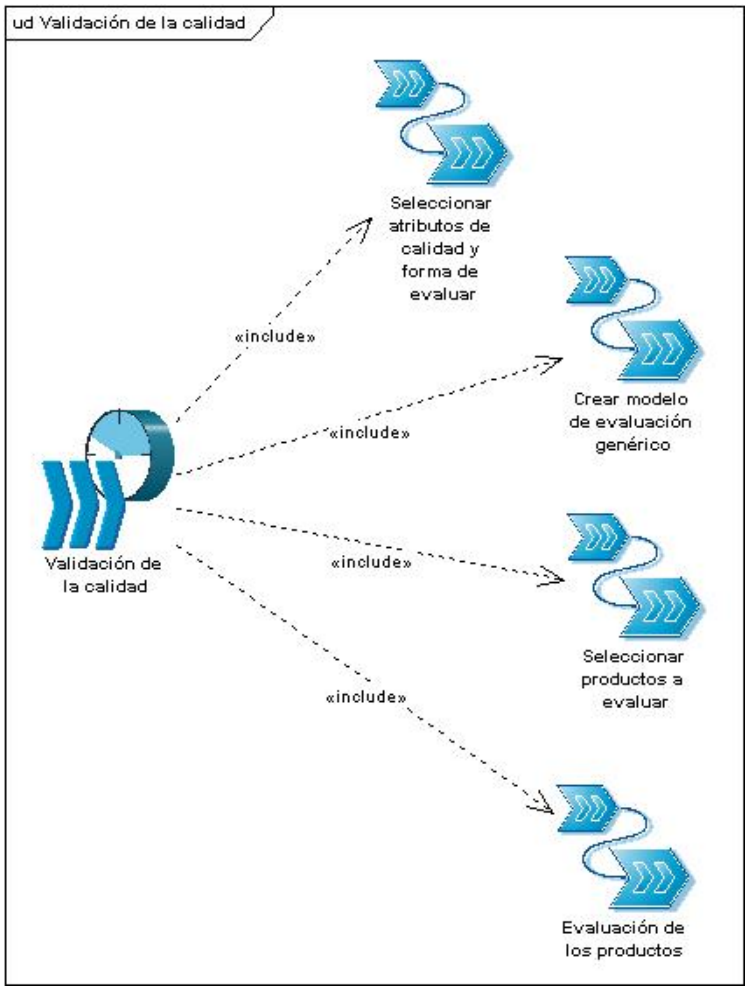


Figura C.14: Diagrama de casos de uso con las definiciones de trabajo que incluye la fase de validación

Se parte del modelo CaFM resultado de la fase de especificar la variabilidad. Primeramente, se seleccionan los atributos de calidad a evaluar y la forma de evaluarlos. Posteriormente se crea un modelo genérico de evaluación siempre y cuando la evaluación sea a nivel de diseño. Luego viene la definición de trabajo de seleccionar los productos o diseños a evaluar que no en todos los casos es obligatorio. Y por último la tarea de evaluación de los productos (ver figura C.15).

A continuación se definen las actividades y los roles que participan en las cuatro definiciones de trabajo de esta fase (ver figuras: C.16, C.17, C.18 y C.19).

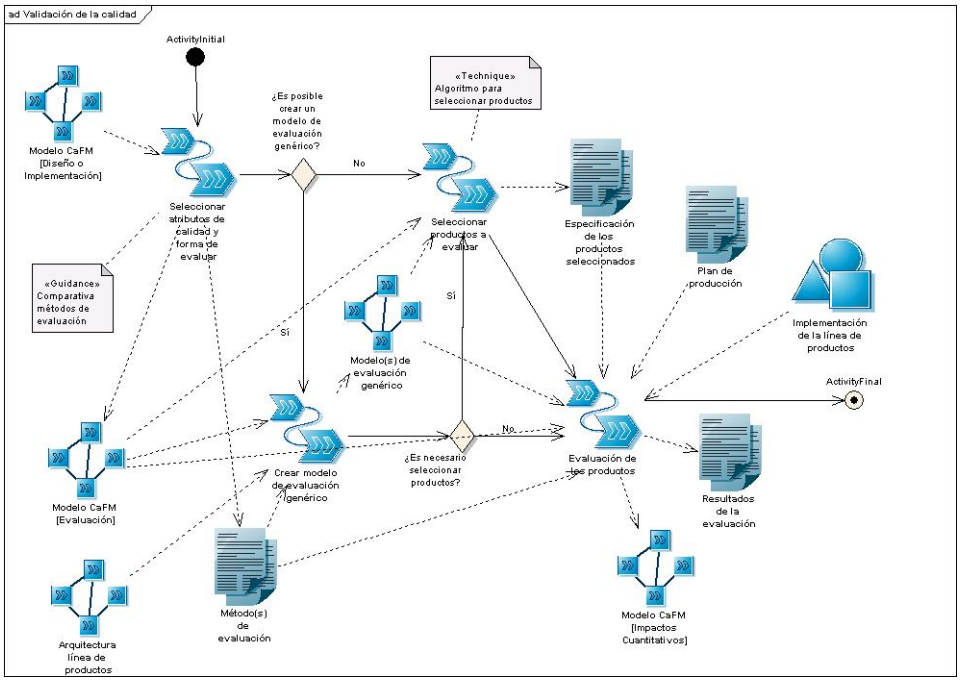


Figura C.15: Diagrama de actividad que describe la fase de validación en términos de definiciones de trabajo y productos de trabajo

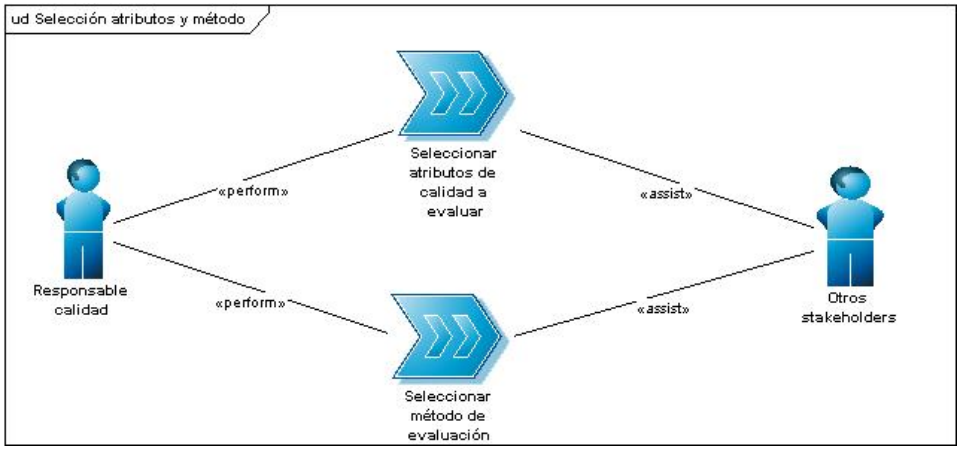


Figura C.16: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Seleccionar atributos de calidad y forma de evaluar"

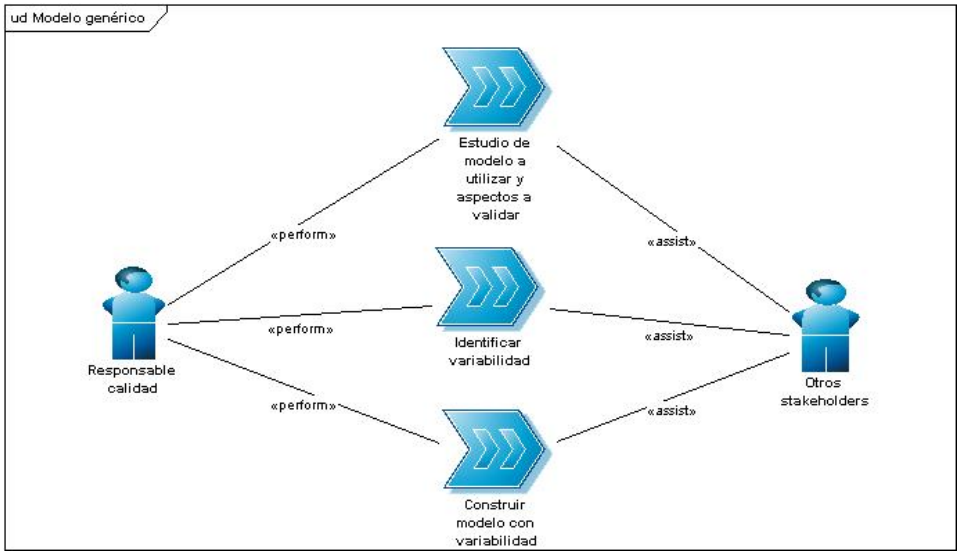


Figura C.17: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Crear modelo de evaluación genérico"

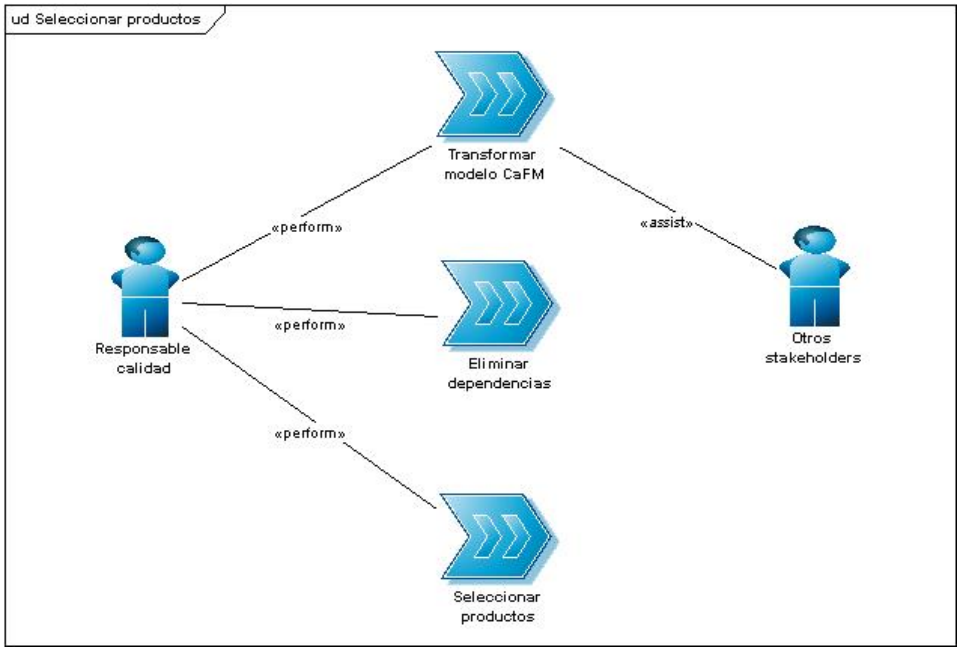


Figura C.18: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Seleccionar productos a evaluar"

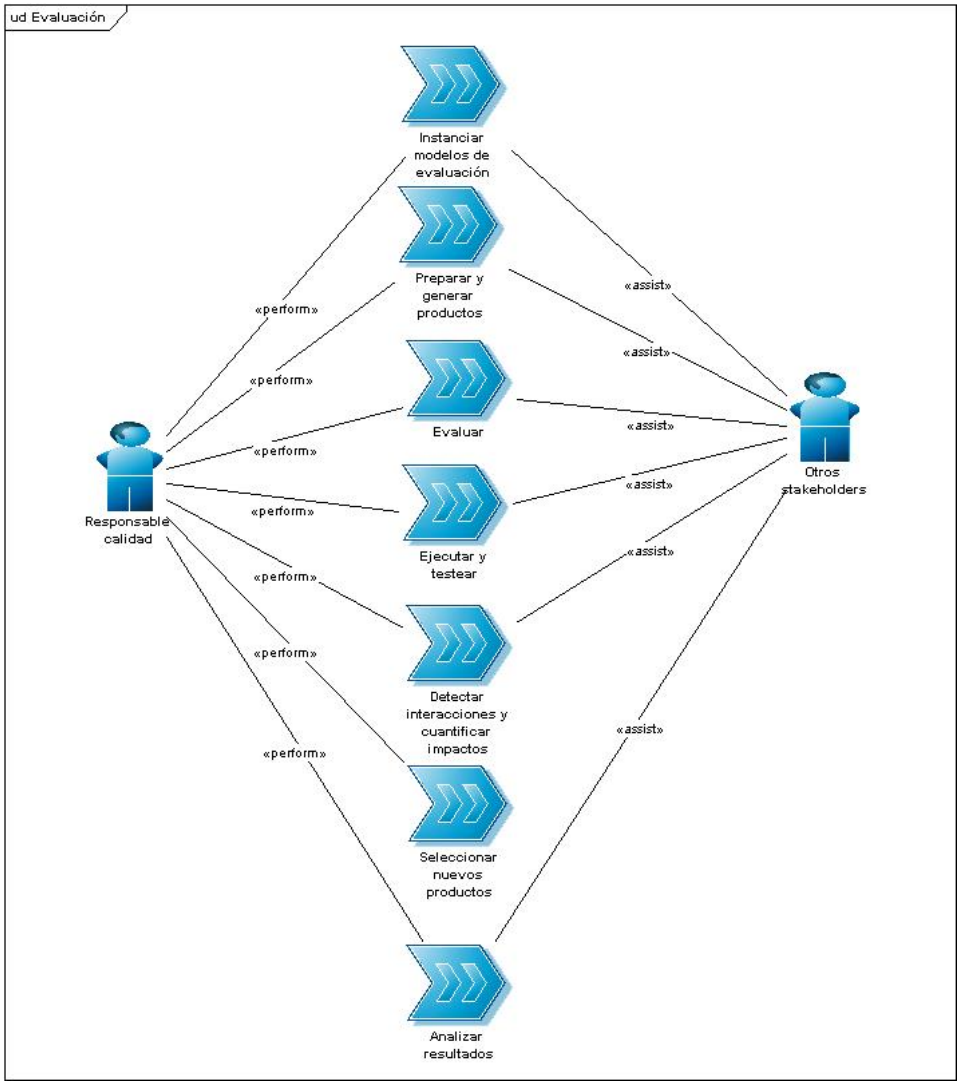


Figura C.19: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Evaluación de los productos"

En la tabla C.2 se muestra un resumen de las actividades de esta fase.

Tabla C.2: Resumen de las actividades de la fase de Validación

Definición de trabajo	**Actividad**	**Descripción de la actividad**	**Rol**
Seleccionar atributos de calidad y forma de evaluar	Seleccionar atributos de calidad a evaluar	Elegir los aspectos de calidad a evaluar, el alcance de la evaluación, el nivel de abstracción (diseño o implementación)	Responsable de calidad
	Seleccionar el método de evaluación	Seleccionar el método de evaluación a utilizar tanto si es un método de evaluación de arquitecturas como un método de testeo	Responsable de calidad
Crear modelo de evaluación genérico	Estudio de modelo a utilizar y aspectos a validar	Estudio de los aspectos que hay que evaluar, del diseño de la línea y del modelo que hay que construir para evaluar esos aspectos	Responsable de calidad
	Identificar variabilidad	Analizar la variabilidad que afecta al modelo y determinar si un modelo es suficiente o serán necesarios varios	Responsable de calidad
	Construir modelo con variabilidad	Construir el modelo de evaluación con la variabilidad identificada	Responsable de calidad
Seleccionar productos a evaluar	Transformar modelo CaFM	Transformar el modelo CaFM en un conjunto de características normalizadas	Responsable de calidad
	Eliminar dependencias	Eliminar las dependencias entre características	Responsable de calidad
	Seleccionar productos	Seleccionar los productos o diseños a evaluar	Responsable de calidad
Evaluación de los productos	Instanciar modelos de evaluación	Instanciar el modelo de evaluación genérico con los datos de los productos concretos	Responsable de calidad
	Preparar y generar productos	Añadir los aspectos necesarios para el posterior análisis y generar los productos ejecutables	Responsable de calidad
	Evaluar	Evaluar los productos utilizando los modelos de evaluación instanciados	Responsable de calidad
	Ejecutar y testear	Ejecutar los productos generados y medir los aspectos de calidad	Responsable de calidad
	Detectar interacciones y cuantificar impactos	Comparar los resultados de la evaluación y detectar las interacciones y cuantificar los impactos	Responsable de calidad
	Seleccionar nuevos productos	Seleccionar nuevos productos a evaluar si es necesario	Responsable de calidad
	Analizar resultados	Analizar los resultados y determinar si se cumplen los atributos de calidad a nivel de línea de productos	Responsable de calidad

Roles involucrados

El rol principal es el del responsable de calidad que es el encargado de velar para que los atributos de calidad requeridos se cumplan en la línea de productos, así

como de tomar decisiones que afecten a la calidad en caso de conflictos. Este rol es muchas veces asumido por el arquitecto software debido a la influencia que tiene la arquitectura en la calidad; sin embargo, en el proceso CaLiPro los atributos de calidad se consideran una entidad de primer orden y se ve necesario un rol que vele por el cumplimiento de estos atributos en la línea.

El responsable de calidad es el encargado de realizar todas las actividades de esta fase con la ayuda de otros *stakeholders*:

- Arquitecto software
- Desarrollador
- Encargados de velar por atributos de calidad específicos: ingeniero de rendimiento, experto en seguridad, etc.
- Testeador
- Encargado de mantenimiento
- Encargados de marketing
- Clientes
- Usuarios finales
- Constructor de aplicaciones
- Expertos de dispositivos
- Etc.

Durante la selección de los atributos de calidad a evaluar, el cliente, el usuario final, el encargado de marketing, el encargado de mantenimiento, etc. pueden ayudar a priorizar los atributos a evaluar. A la hora de determinar el alcance de la evaluación (chequear límites u obtener valores para definir niveles) además de los *stakeholders* mencionados el constructor de aplicaciones y los expertos también pueden aportar información.

Para seleccionar el método de evaluación, el arquitecto software, los expertos en calidad, el testeador, etc. pueden aportar su experiencia con los métodos que conocen.

A la hora de construir el modelo de evaluación genérico así como durante la evaluación, el arquitecto software y los expertos en calidad puede ser de gran utilidad si ya han utilizado el método seleccionado previamente. El testeador, el constructor de aplicaciones y el desarrollador también pueden ayudar durante la evaluación de productos si se realiza a nivel de implementación.

C.5.2.1. Seleccionar atributos de calidad y forma de evaluar

El objetivo de esta definición de trabajo es elegir los aspectos de calidad a evaluar, el alcance de la evaluación, el nivel de abstracción (diseño o implementación) de la evaluación y seleccionar el método de evaluación a utilizar tanto si es un método de evaluación de arquitecturas como un método de testeo (ver figura C.20).

Respecto al alcance de la evaluación, para algunos atributos de calidad caracterizados con escenarios tipo límite (por ejemplo *tiempo de respuesta <1 seg para una operación concreta*) puede ser suficiente con chequear los productos límite (los que van a tener peor tiempo de respuesta) y comprobar que se cumple el escenario. En otros casos, puede que el valor del atributo de cada producto sea importante por lo que hay que cuantificar los impactos para lograr ese valor para todos los productos (pensando ya en una derivación con niveles de calidad).

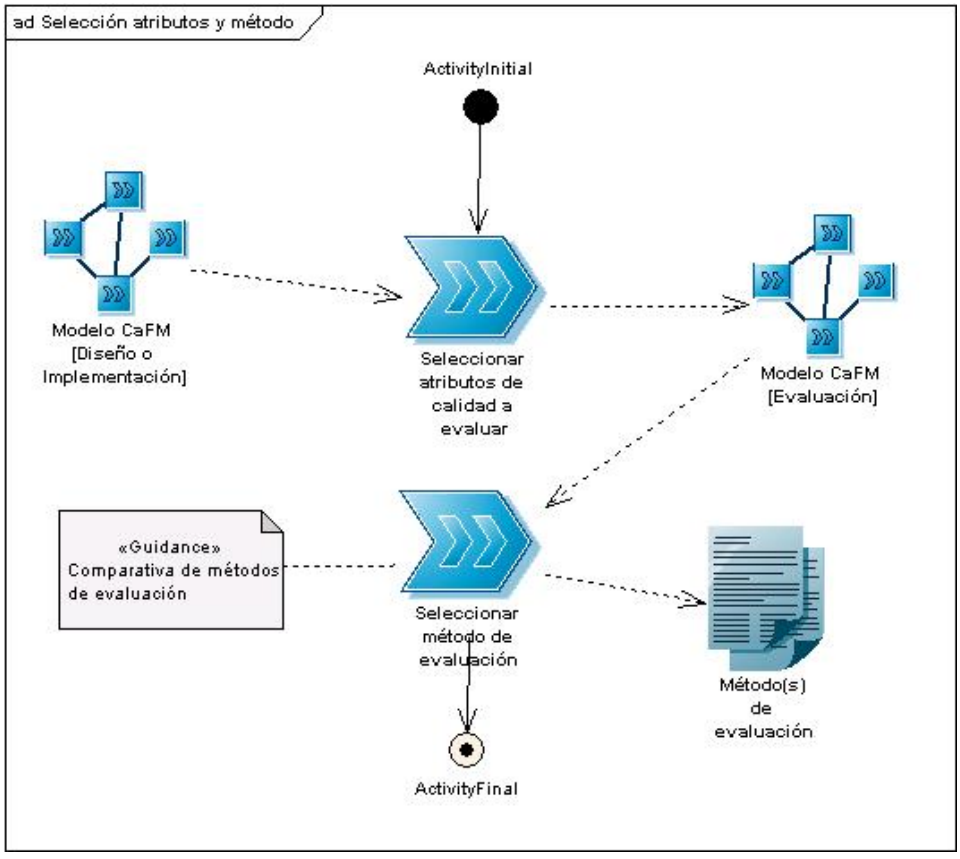


Figura C.20: Diagrama de actividad de la definición de trabajo "Seleccionar atributos de calidad y forma de evaluar"

Entregables

Los entregables de esta definición de trabajo son el modelo CaFM [Evaluación] en el que se han seleccionado los aspectos a evaluar y un documento con el método o métodos de evaluación seleccionados.

Precondiciones

Debe existir un modelo CaFM de nivel de diseño o implementación.

Guías

El responsable de calidad con ayuda de otros *stakeholders* debe definir qué atributos de calidad se van a evaluar ya que en muchos casos no será viable evaluar todos los atributos descritos en el modelo CaFM. La selección dependerá de las prioridades determinadas por los *stakeholders* involucrados en esta tarea: usuarios, clientes, expertos en aspectos de calidad, etc.

También hay que determinar cómo va a ser esa evaluación y qué método se va a utilizar. Es aconsejable disponer de una comparativa de métodos que ayude a seleccionar el más adecuado para cada caso. En nuestro caso disponemos de una comparativa de métodos de evaluación de arquitecturas software (ver anexo E).

Dependencias

Depende de la fase anterior ya que debe existir un modelo CaFM a nivel de diseño o implementación.

Entrada/Salida

La entrada es un modelo CaFM a nivel de diseño o implementación y como salida se obtiene el modelo CaFM [Evaluación] en el que se han seleccionado los aspectos a evaluar y un documento con el método o métodos de evaluación seleccionados.

C.5.2.2. Crear modelo de evaluación genérico

El objetivo de esta definición de trabajo es construir un modelo de evaluación genérico (ver figura C.21).Para ello, primeramente hay que estudiar los aspec-

tos a evaluar, el diseño de la línea y el método de evaluación para definir un primer boceto del modelo que hay que construir para evaluar esos aspectos.

Luego es necesario analizar la variabilidad que afecta a ese modelo y determinar si con un modelo es suficiente o se requieren más.

El siguiente paso es construir el modelo(s) de evaluación con variabilidad y realizar una matriz de trazabilidad en la que se relaciona la variabilidad del modelo y la variabilidad descrita en el modelo CaFM. También se debe analizar si la instanciación del modelo es costosa o no para decidir si es necesario seleccionar ciertos diseños o se pueden evaluar todos los productos.

Entregables

El entregable de esta definición de trabajo es un modelo o modelos de evaluación genéricos que permitan evaluar los atributos de calidad seleccionados previamente.

Precondiciones

Debe existir un modelo CaFM de evaluación y un documento con el método(s) de evaluación seleccionado(s).

Guías

El responsable de calidad con ayuda de otros *stakeholders* debe construir un modelo de evaluación genérico. Entre los *stakeholders* que participan, un stakeholder que conozca y haya usado el método de evaluación con anterioridad será clave: expertos en aspectos de calidad, el arquitecto software, etc.

Además, la información disponible del método de evaluación es también de gran utilidad: libros, artículos, casos de estudio, etc. que describan las fases y técnicas a utilizar.

Dependencias

Depende de la definición de trabajo anterior debido a que los aspectos a evaluar y la forma de evaluar deben estar seleccionados.

Entrada/Salida

Como entrada se tiene un modelo CaFM [Evaluación] con los aspectos a evaluar seleccionados y un documento con el método o métodos de evaluación

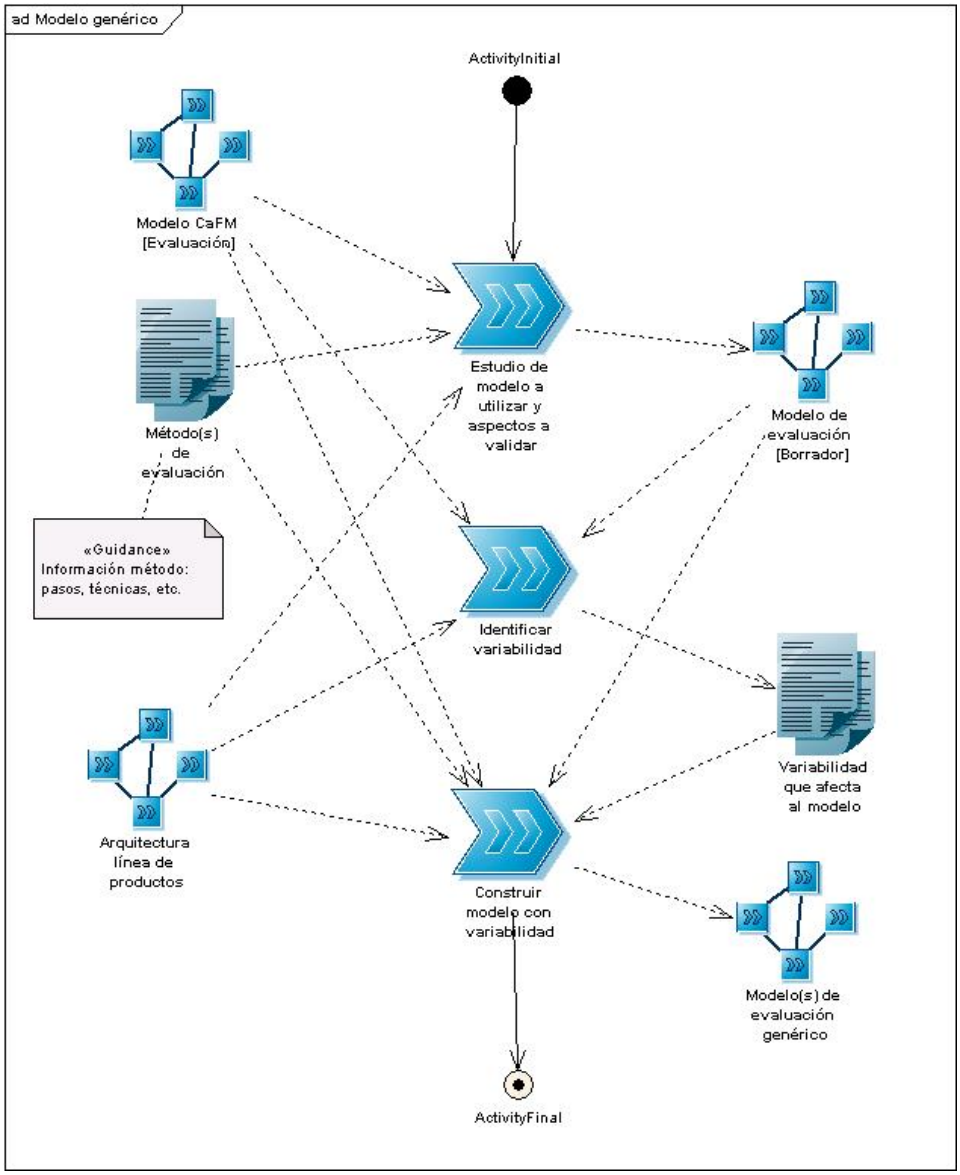


Figura C.21: Diagrama de actividad de la definición de trabajo "Crear modelo de evaluación genérico"

seleccionados. La salida es un modelo o modelos de evaluación genéricos.

C.5.2.3. Seleccionar productos a evaluar

El objetivo de esta definición de trabajo es seleccionar los productos o diseños a evaluar (ver figura C.22). Para ello, primeramente hay que transformar el modelo CaFM en una lista de características normalizadas, es decir, características que tienen solamente variantes alternativas (ver anexo F). A partir de la lista o conjunto de características normalizadas hay que eliminar las dependencias, para que la selección de productos no sea dependiente de las mismas. Por último, hay que seleccionar el mínimo número de productos a evaluar dependiendo del grado de las interacciones que se quieran detectar.

Entregables

El entregable de esta definición de trabajo es la especificación de los productos o diseños seleccionados para evaluar.

Precondiciones

Debe existir un modelo CaFM de evaluación y un modelo de evaluación genérico (si la evaluación es a nivel de diseño y se considera necesaria la selección de productos o diseños).

Guías

El responsable de calidad con ayuda de otros *stakeholders* debe preparar el modelo CaFM y aplicar el algoritmo de selección de productos.

Se han definido guías de cómo transformar el modelo y eliminar las dependencias, así como un algoritmo para seleccionar los productos (ver anexo F para más detalles). Todo ello es aplicable utilizando una herramienta llamada *fmp-CaLiPro*.

Durante la transformación del modelo CaFM hay algunas decisiones a tomar en las que el responsable de calidad puede requerir ayuda de expertos (ver las reglas manuales en el cápitulo 4). El resto de las actividades a desarrollar son mecánicas y con ayuda de la herramienta el encargado de calidad puede realizarlos solo.

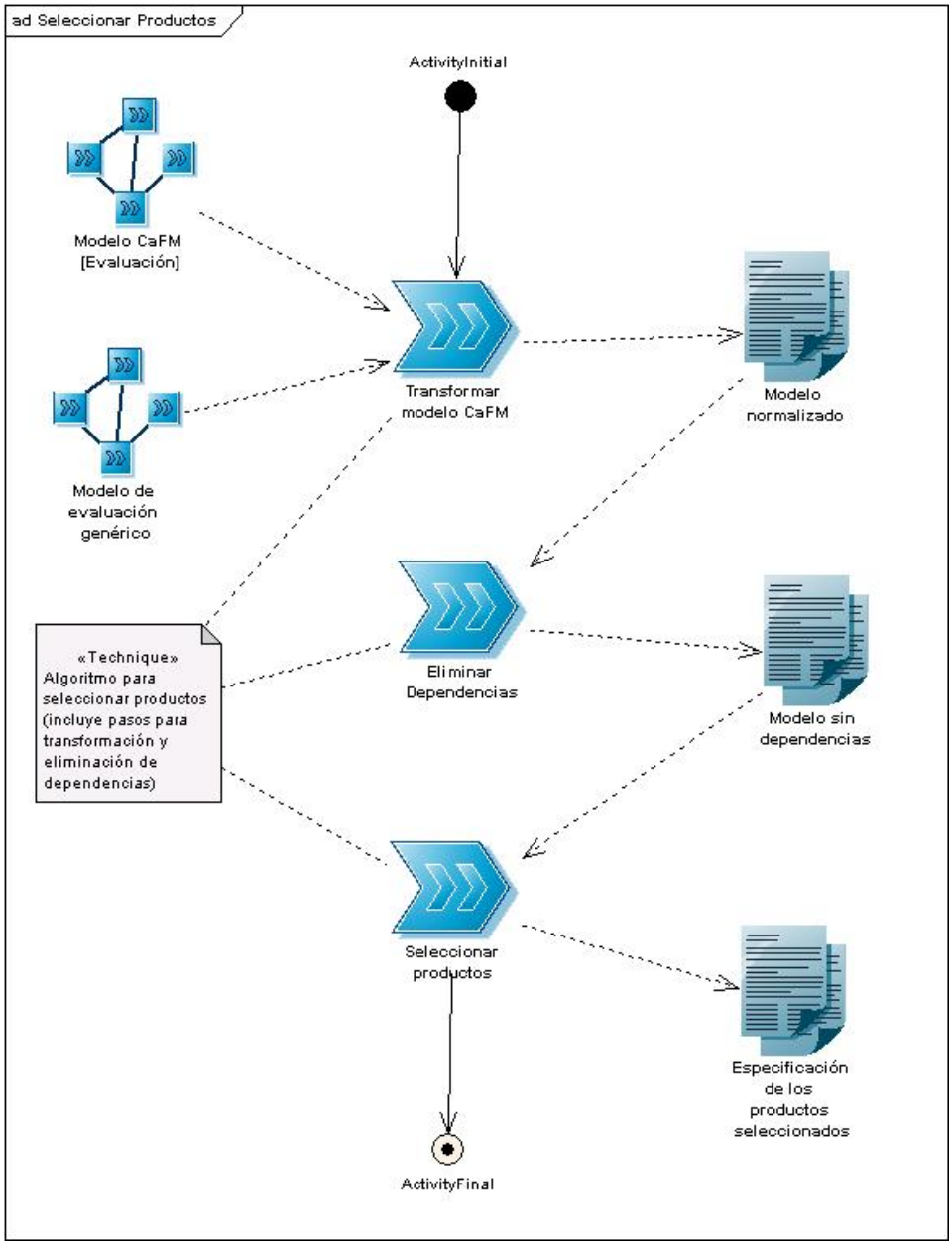


Figura C.22: Diagrama de actividad de la definición de trabajo "Seleccionar productos a evaluar"

Dependencias

Depende de la definición de trabajo "Seleccionar atributos de calidad y forma de evaluar" porque los aspectos a evaluar y la forma de evaluar deben estar seleccionados. También puede depender de la definición de trabajo anterior, es decir, de que exista un modelo(s) de evaluación genérico en los casos que se realice una evaluación a nivel de diseño.

Entrada/Salida

Hay una entrada obligatoria: un modelo CaFM de evaluación y como entrada opcional: un modelo(s) de evaluación genérico (sólo en los casos en los que se construye un modelo de este tipo y se decide evaluar solamente parte de los diseños). La salida es la especificación de los productos o diseños seleccionados para evaluar.

C.5.2.4. Evaluación de los productos

El objetivo de esta definición de trabajo es evaluar o testear los productos seleccionados (ver figura C.23). Dependiendo de si es una evaluación a nivel de diseño o a nivel de implementación, las primeras actividades a realizar son diferentes.

En el caso de una evaluación a nivel de diseño, lo primero a realizar es una instanciación del modelo de evaluación genérico para todos los productos o para los productos seleccionados. Para luego evaluar dichos modelos instanciados siguiendo el método de evaluación.

En el caso de una evaluación a nivel de implementación, primeramente hay que preparar y generar los productos. Preparar consiste en añadir los aspectos necesarios para el posterior análisis como por ejemplo añadir código de medición en los productos. Respecto a la generación de los productos, consiste en componer el producto a partir de los activos de la línea. Esta tarea puede ser más o menos costosa dependiendo de si se utiliza un enfoque de generación automática o no. Una vez que los productos han sido generados, se pueden ejecutar y medir los aspectos a evaluar.

Una vez que se han obtenido los resultados de evaluación o medición hay que comparar los resultados de la evaluación, detectar interacciones entre características y cuantificar los impactos necesarios. Si no es posible cuantificar todos

los impactos requeridos debido a las interacciones detectadas puede ser necesario seleccionar más productos para evaluar y volver a las actividades del inicio.

La última actividad consiste en analizar los resultados para ver si la línea de productos cumple los requisitos de calidad evaluados. Para ello, se chequearán los productos límite respecto a un aspecto de calidad y se comprobará si se cumplen los escenarios definidos.

Entregables

Los entregables de esta definición de trabajo son un documento con los resultados de la evaluación y un modelo CaFM de evaluación con los impactos cuantificados.

Precondiciones

Debe existir una especificación de los productos o diseños seleccionados para evaluar y/o un modelo de evaluación genérico si la evaluación es a nivel de diseño, así como un método de evaluación seleccionado.

Guías

El responsable de calidad con ayuda de otros *stakeholders* debe realizar la evaluación de los productos. Entre los *stakeholders* que participan, un stakeholder que conozca y haya utilizado el método de evaluación con anterioridad será clave: expertos en aspectos de calidad, el arquitecto software, el testeador, etc.

En el caso de la preparación y generación de productos, el desarrollador y el constructor de aplicaciones también pueden ayudar.

Además, la información disponible del método de evaluación es también de gran utilidad: libros, artículos, casos de estudio, etc. que describan las fases y técnicas a utilizar.

Se han definido guías de cómo comparar los resultados, detectar las interacciones y elegir nuevos productos (ver anexo F para más información).

Dependencias

Depende de la definición de trabajo anterior ya que los productos a evaluar deben estar seleccionados y/o de la definición de trabajo "Crear modelo de evaluación genérico", es decir, que exista un modelo(s) de evaluación genérico

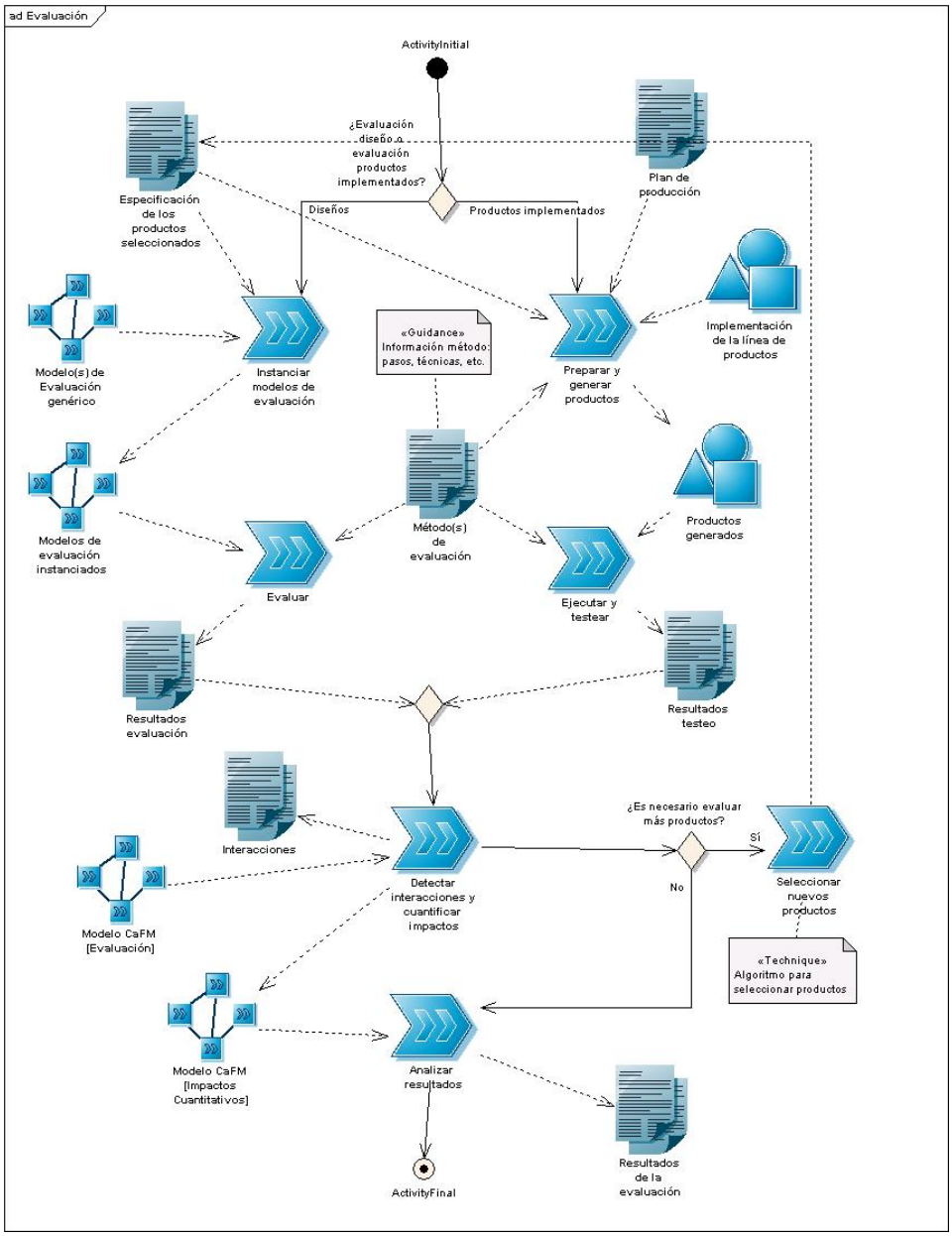


Figura C.23: Diagrama de actividad de la definición de trabajo Evaluación de los productos

en los casos que se realice una evaluación a nivel de diseño.

La forma de evaluar también debe estar seleccionada.

Entrada/Salida

Para una evaluación a nivel de diseño: la entrada es la especificación de los productos o diseños seleccionados para evaluar y/o modelo de evaluación genérico. Así como el método de evaluación.

Para una evaluación a nivel de implementación: la entrada es la especificación de los productos o diseños seleccionados para evaluar, la implementación de la línea y el plan de producción, así como el método de evaluación.

Las salidas son un documento con los resultados de la evaluación y un modelo CaFM de evaluación con los impactos cuantificados.

C.5.3. Fase de Derivación

Los objetivos de esta fase son los siguientes:

1. Añadir características de derivación (niveles) al modelo CaFM para facilitar la derivación teniendo en cuenta aspectos de calidad durante la ingeniería de aplicación.
2. El uso del modelo CaFM de derivación durante la ingeniería de aplicación para permitir una derivación teniendo en cuenta los niveles de calidad.

Esta fase tiene dos definiciones de trabajo (ver figura C.24) muy diferenciados. Uno se encarga de aumentar el modelo CaFM para añadir aspectos de derivación y el otro tiene como objetivo utilizar este modelo y los aspectos o características de derivación añadidos para facilitar la derivación de productos teniendo en mente aspectos de calidad.

Se parte del modelo CaFM resultado de la fase de validación y se aumenta con niveles (características de derivación) que proporcionan información acerca de los niveles de los atributos de calidad. Posteriormente, este modelo CaFM de derivación se puede utilizar en la ingeniería de aplicación como guía durante la selección de características a partir de los requisitos del producto (ver figura C.25).

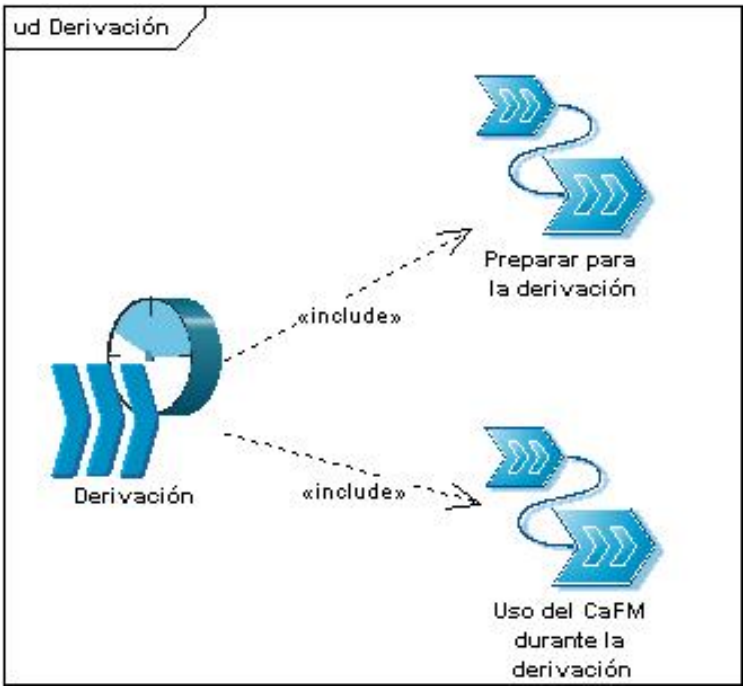


Figura C.24: Diagrama de casos de uso con las definiciones de trabajo que incluye la fase de derivación

A continuación se definen las actividades y los roles que participan en las dos definiciones de trabajo de esta fase (ver figuras:C.26 y C.27).

La tabla C.3 muestra un resumen de las actividades de esta fase.

Tabla C.3: Resumen de las actividades de la fase de Derivación

Definición de trabajo	**Actividad**	**Descripción de la actividad**	**Rol**
Preparar para la derivación	Definir niveles de atributos de calidad	Aumentar el modelo CaFM con características de derivación: niveles o grupos de atributos de calidad	Responsable de calidad
Uso del CaFM durante la derivación	Análisis de requisitos teniendo en cuenta la calidad	Utilizar el modelo CaFM durante la selección de características de un producto para tener en cuenta aspectos de calidad	Constructor de aplicaciones

Roles involucrados

Dos roles son los encargados de realizar las actividades de esta fase: el responsable de calidad y el constructor de aplicaciones. El constructor de aplicaciones es el encargado de instanciar y construir productos a partir de los activos de la línea de productos (arquitectura, implementación...).

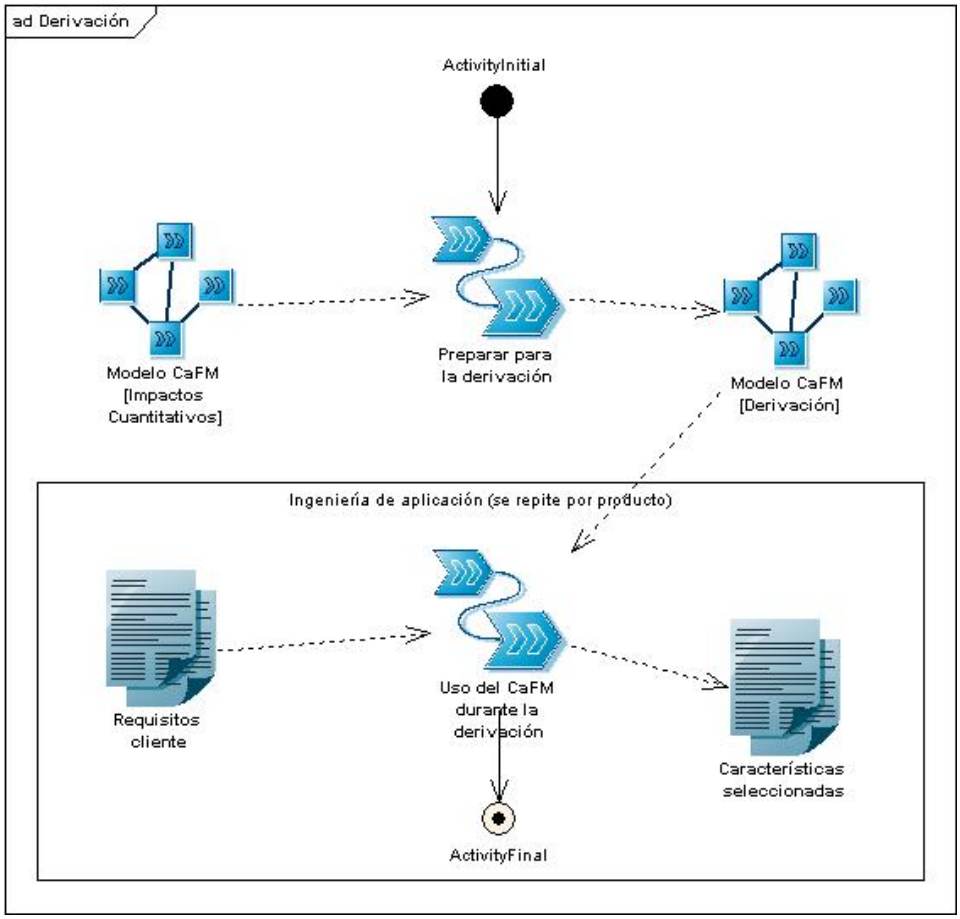


Figura C.25: Diagrama de actividad que describe la fase de derivación en términos de definiciones de trabajo y productos de trabajo

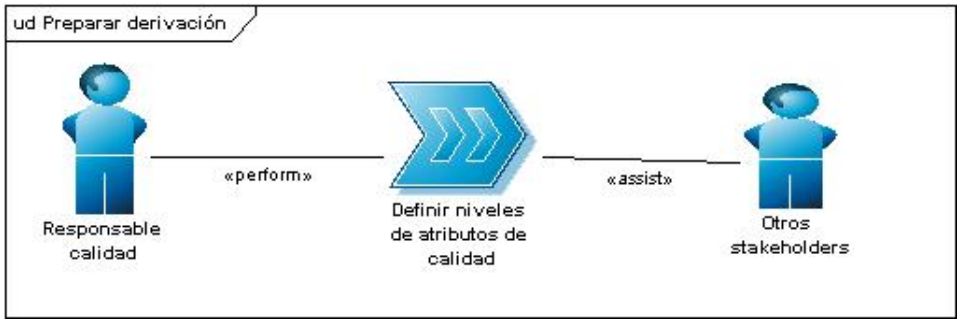


Figura C.26: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Preparar para la derivación"

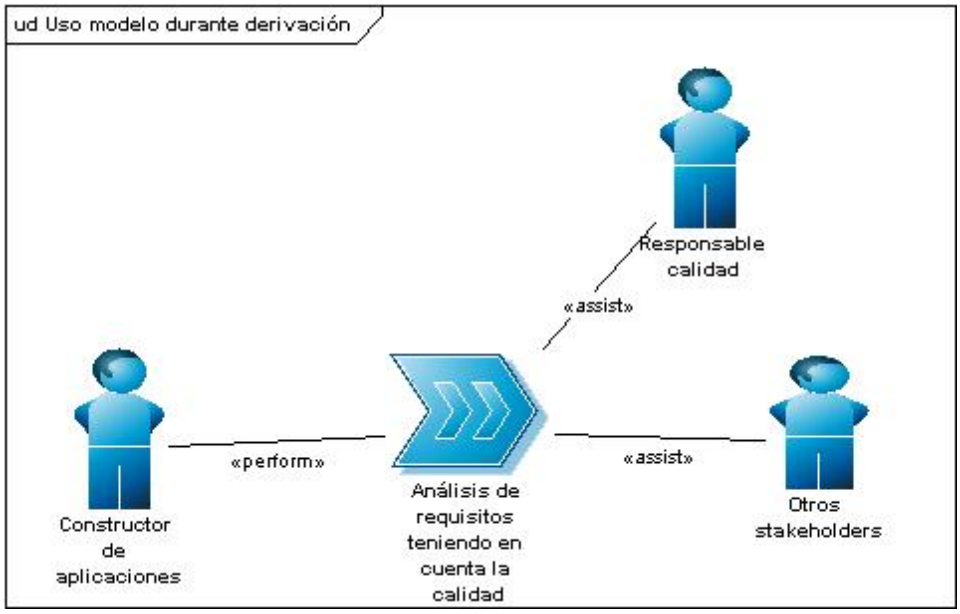


Figura C.27: Diagrama de casos de uso con las actividades y roles de la definición de trabajo "Uso del CaFM durante la derivación"

El responsable de calidad es el encargado de aumentar el modelo CaFM con aspectos de derivación. Y el constructor de aplicaciones es el encargado de usar el modelo CaFM aumentado para realizar una selección de características para un producto teniendo en cuenta aspectos de calidad.

Además de este rol, otros *stakeholders* también pueden estar presentes y ayudar a lo largo de esta fase.

- Encargados de velar por atributos de calidad específicos: ingeniero de rendimiento, experto en seguridad, etc.
- Encargados de marketing
- Clientes
- Usuarios finales
- Etc.

Durante la preparación del modelo para facilitar la derivación, el constructor de aplicaciones, el cliente, el usuario, el responsable de marketing, etc. pueden ayudar a determinar el número de grupos de niveles a realizar para los atributos seleccionados.

Durante el uso del modelo en la derivación, el cliente, el usuario, el responsable de marketing, los expertos de calidad, el responsable de calidad, etc. pueden

ayudar en la toma de decisiones en caso de haber conflictos.

C.5.3.1. Preparar para la derivación

El objetivo de esta definición de trabajo es preparar el modelo CaFM para su posterior uso en la derivación(ver figura C.28). Para ello hay que aumentar el modelo CaFM con niveles de atributos de calidad: se obtienen el valor mínimo y máximo de un aspecto de calidad y se realizan grupos dependiendo de la exactitud deseada (lo que repercute en el número de grupos o niveles de calidad).

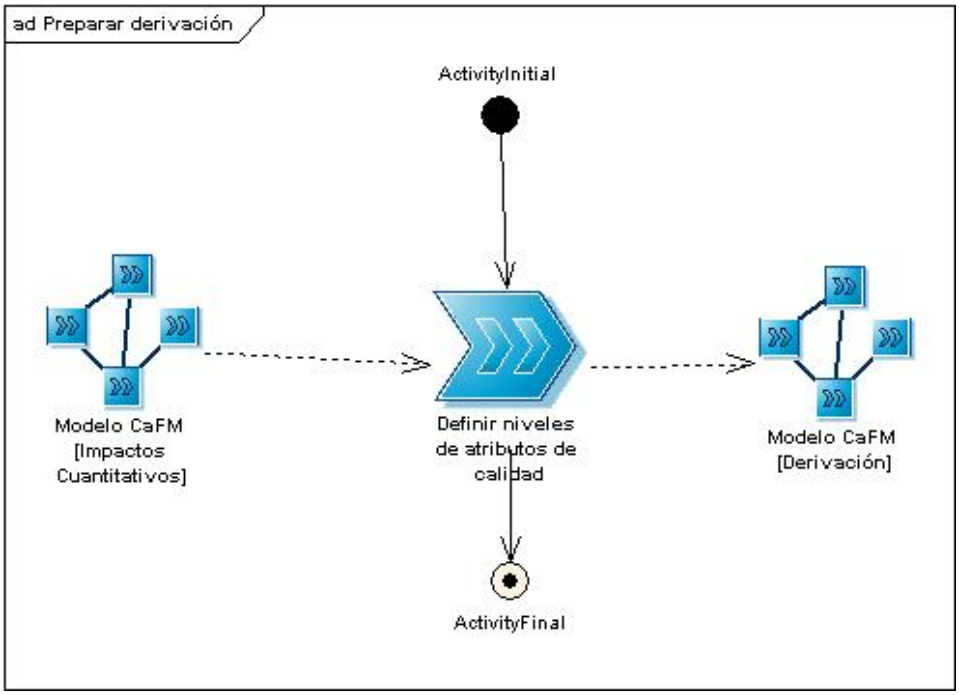


Figura C.28: Diagrama de actividad de la definición de trabajo "Preparar para la derivación"

Entregables

El entregable es un modelo CaFM de derivación.

Precondiciones

Debe existir un modelo CaFM con impactos cuantitativos. Los impactos relacionados con los atributos para los cuales se desean especificar niveles deben estar cuantificados.

Guías

El responsable de calidad debe definir niveles de los aspectos de calidad previamente elegidos. Para especificar el número de grupos deseado puede consultar con otros *stakeholders*: clientes, usuarios, constructor de aplicaciones, expertos en calidad, etc.

Dependencias

Depende de la fase anterior, ya que necesita un modelo CaFM con impactos cuantitativos fruto de la evaluación de productos.

Entrada/Salida

La entrada es un modelo CaFM de evaluación con los impactos cuantificados. La salida es el mismo modelo CaFM aumentado con características de derivación (niveles).

C.5.3.2. Uso del CaFM durante la derivación

El objetivo de esta definición de trabajo es utilizar el modelo CaFM de derivación creado en la definición anterior (ver figura C.29). El modelo CaFM con niveles de atributos de calidad facilita una selección de características teniendo en cuenta los aspectos de calidad a partir de los requisitos de un producto, la información del modelo puede ayudar a ver qué nivel de calidad se obtiene mediante la selección de características funcionales, arquitecturales y de implementación. Los niveles de atributos de calidad también pueden ser seleccionados y obtener información de qué características no se pueden seleccionar o cuáles se deben seleccionar para lograr esos niveles de calidad.

Entregables

El entregable es la selección de características para un producto (seleccionado siguiendo criterios tanto funcionales como de calidad). A partir de esta selección se puede comenzar con la derivación de los productos ejecutables.

Precondiciones

Debe existir un modelo CaFM de derivación con niveles y unos requisitos del producto (o que el cliente participe en esta actividad).

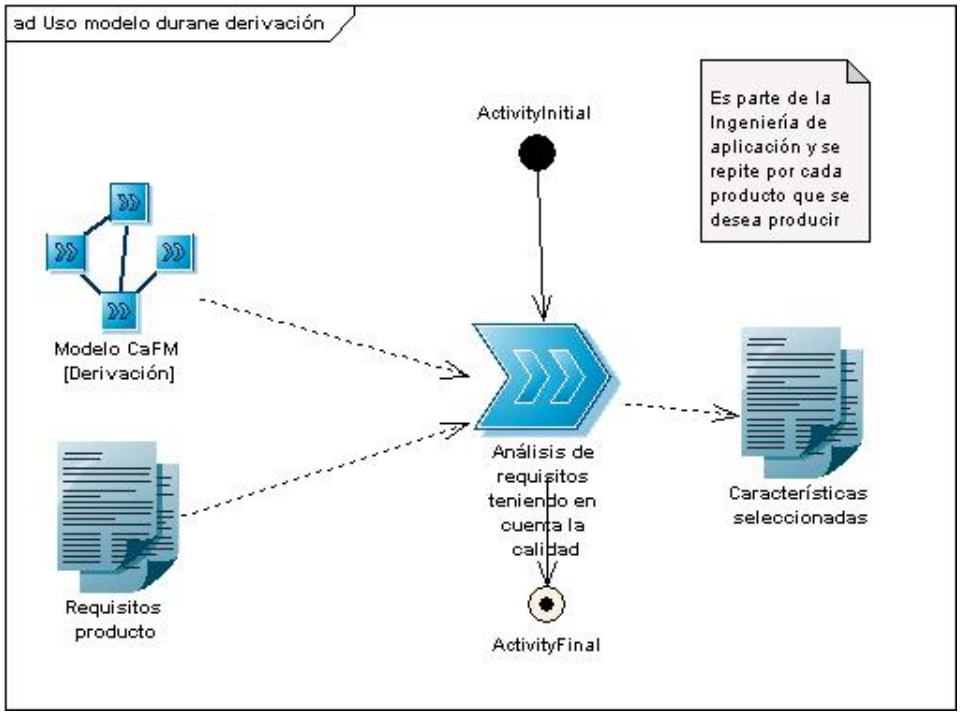


Figura C.29: Diagrama de actividad de la definición de trabajo "Uso del CaFM durante la derivación"

Guías

El constructor de la aplicación debe hacer uso de la información en el CaFM de derivación. El cliente, usuario final, encargado de marketing, etc. también pueden participar definiendo los requisitos del producto.

También es necesario realizar un análisis *trade-off* en el caso de que haya varios atributos de calidad en conflicto y tomar decisiones de qué aspectos priorizar en casos de especificaciones imposibles.

Se ha desarrollado una herramienta (GuiCa) que ayuda durante este proceso.

Dependencias

Depende de la definición de trabajo anterior, ya que necesita un modelo CaFM con niveles.

Entrada/Salida

La entrada es un modelo CaFM de derivación. La salida es un documento con la selección de características para un producto particular.

Anexo D

Extensión del árbol de utilidad

D.1. Introducción

Para especificar los atributos de calidad y su variabilidad se ha integrado una forma de caracterización de atributos de calidad en el modelo de características. En concreto se ha extendido el árbol de utilidad de atributos de calidad, un modelo que se utiliza en las evaluaciones ATAM [Clements 02]. Este documento explica las extensiones propuestas al árbol de utilidad.

Un árbol de utilidad es una estructura de datos que tiene una raíz llamada "*Utility*". Posee tres niveles de nodos debajo de la raíz. Los nodos del primer nivel se llaman atributos de calidad tales como *rendimiento* o *seguridad*. Por debajo de estos nodos se encuentran las elaboraciones o *concerns* - por ejemplo, rendimiento puede ser elaborado como *alto throughput* y *latencia de transición punta a punta corta*. Y el último nivel son las hojas del árbol que describen escenarios que todavía detallan más las elaboraciones. Un árbol de utilidad no es un intento de definir una taxonomía de atributos de calidad rigurosa. Su propósito es obtener una definición de los requisitos de calidad del sistema de una forma práctica [Clements 02], es por lo tanto una forma de caracterizar atributos de calidad.

D.2. *Quality feature tree*: Extensión del árbol de utilidad

El nuevo modelo propuesto se llama Quality feature tree (árbol de características de calidad) ya que es una extensión del árbol de calidad pero teniendo en cuenta las particularidades de las características de calidad y la variabilidad presente en el modelo de características con el que se va a integrar.

En el nuevo modelo Quality feature tree, las características de **calidad** se representan mediante la filosofía del árbol de utilidad y van a tener los subtipos atributos de calidad, elaboraciones y escenarios del árbol de utilidad. Pero no se quiere limitar solamente a los tres tipos propuestos por el árbol de utilidad y tampoco la estructura fija que marca este árbol: debido a que en algunos casos puede ser muy útil tener los tres niveles y en otros casos puede ser suficiente con el primer nivel, o se requieren más de tres debido a la complejidad del atributo de calidad. Por esta razón, se han añadido nuevos subtipos y variabilidad.

Como se muestra en la figura D.1, el quality feature tree completa el árbol de utilidad con nuevos subtipos de refinamientos (métricas, medidas, niveles), estructura flexible (no sólo tres niveles) y variabilidad.

En la tabla D.1 se especifican los aspectos añadidos y las adaptaciones realizadas respecto al árbol de utilidad original.

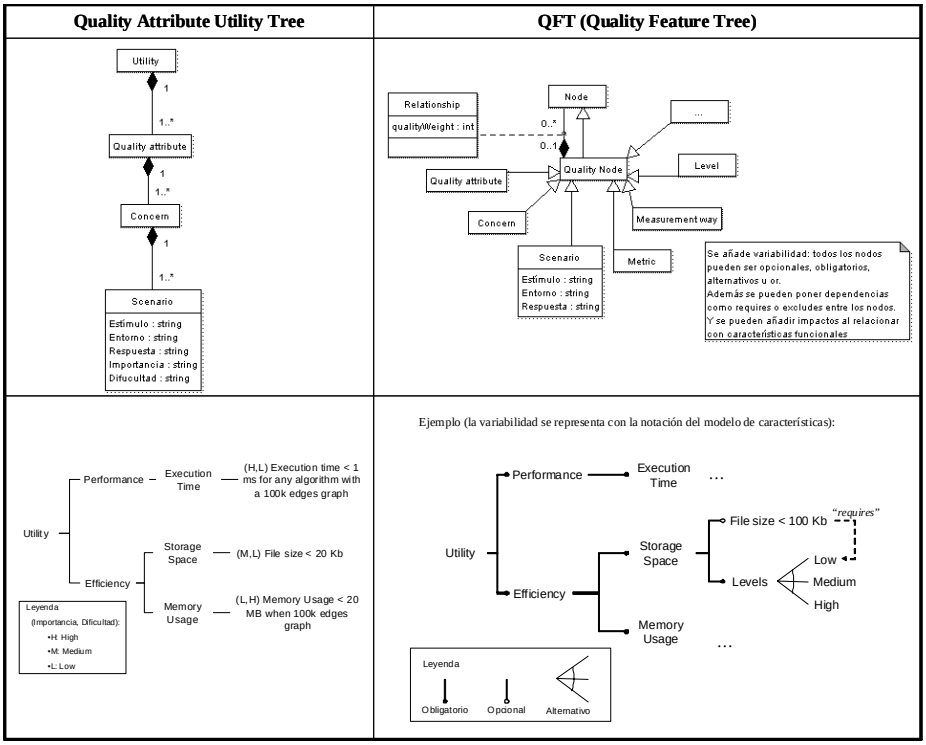


Figura D.1: Árbol de utilidad original versus QFT (*Quality Feature Tree*)

Tabla D.1: Diferencias entre el árbol de utilidad original y la extensión propuesta

	Quality utility tree (original)	**Quality feature tree**
Objetivo	Caracterización de atributos de calidad para su evaluación	Ampliado para facilitar la derivación y especificación de atributos de calidad (además de la evaluación).
Estructura	Estructura de 4 niveles fijo: *Utility* (root), atributos de calidad, refinamiento y escenario.	Estructura flexible: ▪ *Utility* o root integrado con la raíz del modelo de características ▪ Resto de niveles flexible: Pueden ser más o menos de 3 según necesidades
Tipos de nodos	*Utility* (root), atributos de calidad, refinamiento y escenario	Nuevos subtipos de refinamientos: métrica, medida y niveles.
	Escenario	Mismo concepto.
	Atributo de calidad	Mismo concepto. (se recomienda que solo este presente en el primer nivel).
	Refinamiento (*concern*)	Mismo concepto. Pero puede repetirse en varios niveles.Dos subtipos posibles son métrica y medida.
		Métrica: Las métricas son muy útiles durante el refinamiento para permitir la cuantificación de aspectos complejos o no medibles directamente.
		Medida (*measurement way*): Las medidas están relacionadas directamente con la forma de medir los aspectos de calidad. Puede ser necesario introducir diferentes formas de medir.
		Nivel (*level*): Los niveles se introducen pensando en la derivación. Y representan grupos de valores de una característica de calidad.
		Abierto a posibles nuevos tipos de elementos de refinamiento siempre y cuando faciliten la caracterización.
Escenarios anotados con	Estímulo, entorno y respuesta: A la hora de definir los escenarios, lo ideal es que tengan esos tres aspectos.	Lo mismo.
	Importancia	La importancia tal cual no se define pero los pesos permiten especificar el grado en el que un escenario u otra característica ayuda a obtener la característica padre. Y el usuario también marca la importancia al seleccionar o no un escenario. Y también el arquitecto al decidir que aspectos van a contar con niveles.
	Dificultad	No se contempla. La evaluación va estar más dirigida a ver si se cumplen los requisitos de todos los productos.
Variabilidad	No contempla	Variabilidad de los nodos
		Impactos
		Pesos

Anexo E

Comparativa de métodos de evaluación de arquitecturas

Para poder elegir el método de evaluación de arquitecturas software más adecuado para cada caso se ha realizado una comparativa de métodos de evaluación de arquitecturas de un solo sistema. La comparativa no pretende ser exhaustiva, pero sí la base en la que se pueden ir incluyendo métodos existentes, así como nuevos.

Existen muchos métodos de evaluación de arquitecturas software. A fin de poder comparar todos estos métodos se ha definido un *framework* sencillo (ver E.1).

Tabla E.1: *Framework* de clasificación de métodos de evaluación

Elementos del framework	Descripción
Método de evaluación	El método
Nombre completo	El nombre completo que se le da al método
Referencia	¿Dónde ha sido publicado/descrito este método?
Propósito o objetivo	¿Cuál es el objetivo principal del método?
Atributos de calidad	¿Cuántos y qué atributos de calidad cubre el método?
Técnicas o enfoque de evaluación	¿Qué tipo de técnica o enfoque se incluye en el método?
Descripción del método	¿Cómo de detallado está el método?
Validación del método	¿Ha sido el método validado?
Relaciones con otros métodos	Si se basa, usa, extiende o tiene alguna relación con algún otro método

Existen varias comparativas de métodos de evaluación de arquitecturas que han sido muy útiles como fuente de información: [Dobrica 02], [Babar 04] y [Ionita 02].

Además de seguir el *framework* anterior, se ha realizado una división según el atributo que evalúan los métodos. De este modo, primeramente se comparan los métodos generales y posteriormente los métodos específicos para un atributo de calidad (en concreto evolución, rendimiento, fiabilidad y usabilidad). Esta clasificación es muy útil a la hora de elegir el método más adecuado ya que el atributo de calidad será clave a la hora de realizar la elección.

E.1. Métodos generales

Dentro de esta sección se han comparado métodos genéricos: válidos para cualquier atributo. Todos ellos se basan en escenarios para realizar la evaluación:

- **Scenario-Based Architecture Analysis Method (SAAM)** es un método genérico de evaluación que sirve para cualquier atributo de calidad.
- **Architecture Trade-off Analysis Method (ATAM)** es un método multi-atributo que se puede utilizar sobre todo para realizar análisis

trade-off de los atributos.

- **Scenario-based Architecture Reengineering (SBAR)** es otro método multi-atributo.

En la tabla E.2 se comparan los métodos siguiendo el *framework* establecido.

Tabla E.2: Métodos de evaluación de arquitecturas software genéricos

Método de evaluación	SAAM	ATAM	SBAR
Nombre	Scenario-Based Architecture Analysis Method	Architecture Tradeoff Analysis Method	Scenario-based Architecture Reengineering
Referencia	[Clements 02]	[Clements 02]	[Bengtsson 99], [Bosch 00]
Objetivo	Método de análisis de arquitecturas que se basa en escenarios	Analizar varios atributos que pueden estar en conflicto	Método de reingeniería de arquitecturas software para evaluar la habilidad de la arquitectura para lograr los requisitos de calidad
Atributo de calidad	Cualquiera pero sobre todo los relacionados con la evolución: modificabilidad, portabilidad, extensibilidad, integrabilidad...	Cualquiera	Cualquiera
Técnicas de evaluación	Escenarios, análisis de funcionalidad y cambios...	Escenarios, árbol de utilidad, análisis *de trade-offs*,...	Depende del atributo: escenarios, modelos matemáticos, simuladores, razonamiento objetivo
Descripción del método	Muy completo	Muy completo	Muy completo
Validación del método	Validado en varios dominios, dispone de herramienta	Muy validado y utilizado	Varios casos de estudio
Relación con otros métodos	Base de varios métodos: SAAMER, SAAMCS, ASAAM... y predecesor de ATAM	Sucesor de SAAM	-

E.2. Relacionados con la evolución: modificabilidad, mantenibilidad, flexibilidad, etc.

Muchos de los métodos de evaluación de arquitecturas tienen como objetivo evaluar atributos de calidad relacionados con la evolución. A continuación se detallan los métodos que se han comparado (ver también la tabla E.3):

- **SAAMER: Software Architecture Analysis Method for Evolution and Reusability** es un método que extiende SAAM para los atributos de evolución y reutilización.

- **SAAMCS: SAAM for Complex Scenarios** es una extensión de SAAM para analizar la flexibilidad de una arquitectura software.
- **ESAAMI o ISAAMCR: Extending SAAM by Integration in the Domain or Integrating SAAM in Domain-centric and Reuse-based development processes** es una extensión de SAAM para evaluar la arquitectura respecto a la reusabilidad y la evolución.
- **Architecture Level Prediction of Sofware Maintenance (ALPSM)** es un método para la predicción de la mantenibilidad del software durante el diseño de la arquitectura.
- **Architecture-level modifiability analysis (ALMA)** es un método para analizar la modificabilidad de una arquitectura.

Tabla E.3: Métodos de evaluación para atributos relacionados con la evolución

Método de evaluación	SAAMER	SAAMCS	ESAAMI / ISAAMCR	ALPSM	ALMA
Nombre	SAAM for Evolution and Reusability	SAAM for Complex Scenarios	Extending SAAM by Integration in the Domain or Integrating SAAM in Domain-centric and Reuse-based development processes	Architecture Level Prediction of Sofware Maintenance	Architecture-level modifiability analysis
Referencia	[Lung 97]	[Lassing 99]	[Molter 99]	[Bengtsson 99]	[Bengtsson 04]
Objetivo	Evaluar las arquitecturas para la reutilización y la evolución	Método de análisis de la flexibilidad en el dominio de sistemas administrativos	Analizar la flexibilidad para reutilizar	Predecir mantenibilidad	Analizar el impacto del cambio, predecir esfuerzo de mantenimiento
Atributo de calidad	Reusabilidad / evolución	Flexibilidad	Flexibilidad, reusabilidad	Mantenibilidad	Modificabilidad
Técnicas de evaluación	Modelado de información y escenarios	Escenarios (complejos), instrumento de medición que incluye diferentes factores	Escenarios	Escenarios	Depende del objetivo del análisis
Descripción del método	Razonable	Razonable	Razonable	Razonable	Razonable
Validación del método	Un caso de estudio	Varios casos de estudio en el dominio de los sistemas administrativos	No se ha publicado	Algunos ejemplos	Validado en diferentes dominios
Relación con otros métodos	Extensión de SAAM	Basado en SAAM	Basado en SAAM	-	Combina los métodos ALPSM y SAAMCS

E.3. Rendimiento (*Performance*)

Existen varios métodos de evaluación de rendimiento que tienen en cuenta aspectos de tiempo y recursos del sistema. La mayoría se basan en las siguientes técnicas: redes de encolado (*Queuing network models*), redes de petri, álgebras de proceso (*process algebras*) y en la teoría de planificación (*scheduling*) de sistemas de tiempo real [Klein 93].

Los QNM se construyen con centros de servicio y colas. Los centros de servicio proveen servicios y tienen una cola asociada. Existen diversas extensiones y combinaciones de estos modelos como por ejemplo *Layered Queuing Network modeling* (LQN), QNM junto con *Layered Transition System* (LTS).

Las redes petri describen el sistema utilizando sitios, transiciones y tokens; varias extensiones se han desarrollado para añadir información de tiempo: CPN (*Colored Petri Nets*), STPN (*Stochastic Timed Petri Net*), GSPN (*Generalized Stochastic Petri Net*), etc.

Las álgebras de proceso son lenguajes algebraicos que se utilizan para describir y verificar propiedades funcionales de sistemas concurrentes y distribuidos. Los SPA (stochastic process algebras) son extensiones de estas algebras para permitir el análisis de propiedades de rendimiento.

La teoría de planificación (*scheduling*) de sistemas de tiempo real se utiliza para analizar la *schedulability* garantizando que los *deadlines* críticos se cumplen incluso en los peores casos.

En la clasificación se explican con más detalle los métodos de evaluación de rendimiento que se han considerado más importantes (ver tabla E.4):

- **PASA (Performance assessment of software architectures)** [Williams 02] es un método para evaluar el rendimiento de las arquitecturas software. Utiliza los principios y técnicas de la ingeniería de rendimiento software [Smith 02].
- **RMA (Rate-monotonic analysis)** es una conjunto de técnicas cuantitativas que se utilizan para entender, analizar y predecir el comportamiento de tiempo de los sistemas [Klein 93].
- **LQN (Layered Queuing Network modeling)**es una extensión de Queuing Network Model (QNM).
- **CPN (Colored Petri Nets)** es una extensión de las redes de petri para evaluar el rendimiento.

Tabla E.4: Métodos de evaluación para rendimiento

Método de evaluación	RMA	PASA	Enfoque LQN	Enfoque CPN
Nombre	Rate-monotonic analysis	Performance assessment of software architectures	Layered Queuing Network modeling	Enfoque Colored Petri Nets
Referencia	[Klein 93]	[Williams 02]	[Petriu 00]	[Wells 02]
Objetivo	Garantizar que los *deadline* críticos se cumplen incluso en las peores situaciones. También se puede usar para evaluar y comparar arquitecturas.	Evaluación del rendimiento con el objetivo de descubrir potenciales problemas en un desarrollo nuevo o decidir si migrar a una nueva arquitectura	Analizar tiempos de respuesta, *throughput*, utilización de servidores con diferentes tipos de peticiones y retrasos de el encolado	Evaluar mecanismos y políticas alternativas en arquitecturas de ejecución
Atributo de calidad	*Schedulability*	Rendimiento (*Performance*)	Rendimiento (*Performance*)	Rendimiento (*Performance*)
Técnicas de evaluación	Basado en la teoría *rate monotonic scheduling* (RMS) que se aplica sobre modelos de tareas	Análisis basado en el modelo de ejecución del software y del sistema derivados a partir de los diagramas de secuencia	Modelos LQN (Layered Queuing Network) que se transforman a partir de diagramas UML, mapas de casos de uso o patrones arquitectónicos	La arquitectura en UML es mapeada a la arquitectura de ejecución como un modelo CPN que se puede simular
Descripción del método	Muy completo	Muy completo	Muy completo	Razonable
Validación del método	Muy utilizado. Existen herramientas.	Validado en varios dominios, dispone de herramienta	Utilizado en varios dominios	Un caso de estudio en software para teléfonos móviles
Relación con otros métodos	-	-	Extensión de enfoque QNM	Extensión de las redes de petri. Existen otros métodos que se basan en otro tipos de extensiones de estas redes

Existen diversas comparativas de métodos de evaluación de rendimiento que pueden utilizarse como fuente de más información como por ejemplo [Purhonen 05], [Balsamo 03a], [Balsamo 03b], [Balsamo 01] y [Herzog 01].

E.4. Fiabilidad (*Reliability*)

Dentro de los métodos de predicción de fiabilidad, *Goseva-Popstojanova y Trivedi* [Goseva-Popstojanova 01] clasifican los enfoques en basados en estados, basados en caminos y modelos aditivos.

Los métodos basados en estados utilizan las probabilidades de transferencia de control entre componentes para estimar la fiabilidad; mientras que los métodos basados en caminos se basan en los caminos de ejecución posibles. Los modelos aditivos calculan la intensidad de fallos de un software compuesto a partir de la intensidad de fallos de los componentes.

Dentro de los modelos basados en estados, los modelos pueden ser compuestos o jerárquicos. Los modelos compuestos combinan el comportamiento de arquitectura y fallos en un modelo único; mientras que los modelos jerárquicos separan la representación de la arquitectura software de la representación del comportamiento de fallo de los componentes.

Los métodos que se han comparado son los siguientes (ver tabla E.5):

- **Enfoque de Reussner et al (Reliability prediction for component-based software architectures)** es un enfoque de modelos analíticos compuestos basados en estados.
- **Enfoque de Rodrigues et al. (Using scenarios to predict the reliability of concurrent component-based software systems)** es un enfoque basado en escenarios. Se clasifica dentro de los enfoques basados en estados y modelos compuestos.
- **SBRA (Scenario-Based Reliability Analysis)** utiliza un modelo CDG (Component Dependency Graph) y es un modelo basado en caminos.
- **Enfoque de Cortellessa et al. (Early reliability assessment of UML based software models)** es un modelo basado en anotaciones o extensiones de la notación UML.
- **Enfoque de Grassi (Architecture-based dependability predic-**

tion for service-oriented computing) se centra en la fiabilidad de servicio y analiza diferentes alternativas arquitecturales.

- **Enfoque de Wang et al. (An architecture-based software reliability model)** se centra en discutir estilos arquitecturales.

Tabla E.5: Métodos de evaluación para fiabilidad

Método de evaluación	SBRA	Enfoque de Reussner et al	Enfoque de Rodrigues et al.	Enfoque de Cortellessa et al.	Enfoque de Grassi	Enfoque de Wang et al.
Nombre	Scenario-Based Reliability Analysis	Reliability prediction for component-based software architectures	Using scenarios to predict the reliability of concurrent component-based software systems	Early reliability assessment of UML based software models	Architecture-based dependability prediction for service-oriented computing	An architecture-based software reliability model
Referencia	[Yacoub 99]	[Reussner 03]	[Rodrigues 05]	[Cortellessa 02]	[Grassi 04]	[Wang 99]
Objetivo	Analizar la fiabilidad de aplicaciones basadas en componentes en función de sus componentes e interfaces	Predecir la fiabilidad del sistema mediante análisis composicional de perfiles de uso y fiabilidad de los componentes del entorno	Predecir la fiabilidad de un sistema software teniendo en cuenta la estructura	Predecir la fiabilidad del sistema basándose en los ratios de fallos de componentes y conectores	Predecir la confiabilidad (*dependability*) incluyendo la fiabilidad de un ensamblado de servicios existentes y desarrollados independientemente	Predecir la fiabilidad de sistemas heterogéneos de acuerdo a la fiabilidad de cada componente, el perfil operacional y la arquitectura
Atributo de calidad	Fiabilidad (*Reliability*)	Fiabilidad (*Reliability*)	Fiabilidad (Reliability)	Fiabilidad (*Reliability*)	Confiabilidad (*dependability*) que incluye la fiabilidad	Fiabilidad (*Reliability*)
Técnicas de evaluación	Modelo probabilístico basado en escenarios: CDG (*Component Dependency Graph*) que representa los componentes, su fiabiliidad, la fiabilidad de las uniones e interfaces y las transiciones y sus probabilidades.	Cadenas Markov	*High-level message sequence chart* (HMSC) anotado con probabilidades de transición de escenarios derivados del perfil operacional del sistema. Y el modelo de fiabilidad del software orientado a usuario de Cheung [Cheung 80] (cadenas Markov)	Anotaciones en los diagramas de casos de uso, secuencia y despliegue utilizando extensiones UML para incorporar información acerca del uso esperado del sistema (perfil operacional) y las distribuciones de probabilidades de fallos de los componentes y conectores	Flujos de peticiones modeladas con cadenas de Markov de tiempo discreto	Cademas Markov
Descripción del método	Razonable	Razonable	Razonable	Razonable	Razonable	Razonable
Validación del método	Un caso de estudio para ilustrar la aplicación del método	Evaluación empírica realizada por los autores	Evaluación empírica realizada por los autores	Evaluación experimental realizada por los autores	Un ejemplo	Validado con experimentos

Una comparativa de métodos de evaluación de fiabilidad a nivel arquitectural muy completa es la de Immonen y Niemelä [Immonen 08]. También hay alguna más como [Goseva-Popstojanova 01].

E.5. Usabilidad

Respecto a la usabilidad la mayoría de métodos de evaluación están orientados a sistemas implementados. Para sistemas sin desarrollar existe un método: SALUTA (*Scenario based Architecture Level UsabiliTy Assessment*). Este método está basado en escenarios y evalúa si una arquitectura cumple los requisitos de usabilidad (ver tabla E.6)

Tabla E.6: Método de evaluación para usabilidad

Método de evaluación	SALUTA
Nombre	Scenario based Architecture Level UsabiliTy Assessment
Referencia	[Folmer 03]
Objetivo	Evaluar si la arquitectura cumple los requisitos de usabilidad
Atributo de calidad	Usabilidad
Técnicas de evaluación	Escenarios y perfiles de uso
Descripción del método	Razonable
Validación del método	Caso de estudio industrial
Relación con otros métodos	-

Relacionado con la evaluación de la usabilidad, Golden et al [Golden 05] han realizado un experimento control para evaluar la utilidad de porciones del patrón USAP (*Usability-Supporting Architectural Pattern*) a la hora de modificar el diseño de arquitecturas software para soportar aspectos de usabilidad específicos.

Anexo F

Selección de productos y detección de interacciones

F.1. Introducción

A *FeatureList* (FL) es un conjunto de características normalizadas o *NormalizedFeature* (NF). El número de características es n, siendo el conjunto de 1 a n (*Number of Normalizad Features*: NFF) el que indica los posibles subíndices de las características normalizadas.

$$\begin{gathered} FL = \{NF_1, NF_2, ..., NF_n\} \\ NFF = \{1, 2, ..., n\} \\ \forall_{x \in NFF} NF_x \in FL \end{gathered} \tag{F.1}$$

Un *NormalizedFeature* (NF) es un conjunto de variantes. El número de variantes es m, siendo el conjunto de 1 a m (*Number of Variants*: NV) el que indica los posibles subíndices de las variantes.

$$\begin{gathered} NF_x = \{V_{x1}, V_{x2}, ..., V_{xm}\} \\ NV_x = \{1, 2, ..., m\} \\ \forall_{x \in NFF} \forall_{y \in NV_x} V_{xy} \in NF_x \end{gathered} \tag{F.2}$$

Un *ProductList* (PL) es un conjunto de productos. El número de productos es s, siendo el conjunto de 1 a s (*Number of Products*: NP) el que indica los posibles subíndices de las productos.

$$\begin{gathered} PL = \{P_1, P_2, ..., P_s\} \\ NP = \{1, 2, ..., s\} \\ \forall_{x \in NP} P_x \in PL \end{gathered} \tag{F.3}$$

Un producto (P) es un conjunto de variantes, seleccionando solamente una de las posibles variantes en cada una las características del FL.

$$P = \{V_{xy} | \forall_{x \in NFF} \exists!_{y \in NV_x} : V_{xy} \in NF_x\} \tag{F.4}$$

Para todo x (siendo x el número de característica de la lista de características), existe uno y sólo un y, siendo y (el número de variante de la característica x).

F.2. Interacciones o impactos

Una interacción de características ocurre cuando una o más características modifican o influyen otras características [Liu 05]. A lo largo de la tesis, cuando se hace referencia a interacción de características se refiere a interacciones de características que causan variación en características de calidad (impactos). Los impactos son un caso particular de interacción de características que representan la variación indirecta (la variabilidad funcional que causa variación en los requisitos de calidad). Una vez que los requisitos tanto funcionales como de calidad están representados en el modelo de características extendido, se definen impactos de forma explícita entre las variantes funcionales, arquitectónicas y de implementación y los requisitos de calidad. Estos impactos pueden ser cualitativos o cuantitativos:

- Impactos cualitativos: Estos impactos se definen en un primer paso y son dependientes del diseñador y expertos. Los impactos pueden ser: ++,+,+-,-,- -.
- Impactos cuantitativos: Son el resultado de evaluaciones de diseños de ciertos productos o del testeo o medición de estos productos.

El **impacto** denota cómo cambia la característica de calidad q a consecuencia de un cambio en la característica f_i. El cambio en la característica de calidad será un cambio en su valor (normalmente en su cantidad) mientras que el cambio en la característica f será cambiar de una variante a otra variante de la característica.

$$Impacto = \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})} \tag{F.5}$$

Se considerará la característica f_i una característica normalizada con variantes alternativas $v_{ij} \in f_i$, $v_{ik} \in f_i$ y $j \neq k$. Por ejemplo: $f_1 = \{v_{11}, v_{12}, v_{13}\}$. El valor del atributo q para el producto 1 sería el siguiente, siendo q_{base} el valor de q del producto con el mínimo valor de q:

$$q_{prod_1} = q_{base} + \sum_{i=0}^{i=NumFeatures} \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})} \tag{F.6}$$

El grado de interacción puede variar dependiendo del número de características involucradas. Así un grado 1 es cuando una característica causa una variación en una característica de calidad, es decir, un impacto simple: $Interact(f_1; q_{f_1})$. Un impacto o interacción de grado n se da cuando n características interaccionan con una característica de calidad, es un impacto de grupo, ya que el valor del impacto depende del valor de las n características involucradas y no es la suma de los impactos individuales: $Interact(f_1, f_2, f_3, ..., f_n; q_{f_1})$.

Una interacción grado n implica interacciones grado n-1 entre las características implicadas en la interacción grado n. En este sentido, no se consideran las interacciones 3-way [Kawauchi 03]: 3 características que interaccionan sin que haya interacciones dos a dos, porque son muy improbables especialmente en este contexto.

Si el cambio en el valor de la característica q cuando cambian dos características no es la suma de los impactos individuales (el cambio que causa una característica más el cambio que causa la otra característica), hay una interacción grado 2 (o mayor): $Interact(f_i, f_l; q)$

$$si \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})\Delta f_l(v_{lm} \rightarrow v_{ln})} \neq \frac{\Delta q}{\Delta f_i(v_{ij} \rightarrow v_{ik})} + \frac{\Delta q}{\Delta f_l(v_{lm} \rightarrow v_{ln})} \tag{F.7}$$

Si hay interacciones de grado 2 o superior, hay que definir impactos grupales o impactos individuales pero condicionados (a los valores del resto de características con los que interacciona, representados con el signo de la derivada parcial). Un impacto condicionado es el cambio en q a consecuencia de un cambio en f_i donde se cambia de la v_{ij} a v_{ik} manteniendo otras características con las que interacciona f_i constantes $(v_{lm}, v_{op}, ...)$.

$$Impacto = \frac{\partial q(v_{lm}, v_{op}, ...)}{\partial f_i(v_{ij} \rightarrow v_{ik})} = q_{f_i} = \partial_{f_i(v_{ij} \rightarrow vik)} q(v_{lm}, v_{op}, ...) \quad \text{(F.8)}$$

En el siguiente ejemplo se muestran dos ejemplos, uno con impactos de grado 1 y otro con impactos de grado 2. En la parte superior de la tabla se detallan los valores de los impactos que se podrían obtener tras una evaluación y en la parte inferior como se puede obtener el valor para q de cada uno de los productos. Por ejemplo, el $Prod_4 = \{b, d, g\}$ en el primer ejemplo tendría un valor de 180 a partir de la suma de la base y los impactos correspondientes de cambiar de a a b, de c a d y de f a g. En el segundo ejemplo, el valor del mismo producto sería 190: la suma de la base más los impactos por cambiar de a a b siendo la característica dos c, de c a d siendo la característica uno b y de f a g.

Características	Productos	
$f_1 = \{v_{11}, v_{12}\} = \{a, b\}$	$Prod_1 = \{a, c, f\}$	$Prod_5 = \{a, c, g\}$
$f_2 = \{v_{21}, v_{22}\} = \{c, d\}$	$Prod_2 = \{b, c, f\}$	$Prod_6 = \{b, c, g\}$
$f_3 = \{v_{31}, v_{32}\} = \{f, g\}$	$Prod_3 = \{a, d, g\}$	$Prod_7 = \{a, d, f\}$
	$Prod_4 = \{b, d, g\}$	$Prod_8 = \{b, d, f\}$

Ej. impacto. grado 1
$\frac{\Delta q}{\Delta f_1(v_{11} \rightarrow v_{11})} = \frac{\Delta q}{\Delta f_1(a \rightarrow b)} = 50$
$\frac{\Delta q}{\Delta f_2(v_{21} \rightarrow v_{21})} = \frac{\Delta q}{\Delta f_2(c \rightarrow d)} = 80$
$\frac{\Delta q}{\Delta f_3(v_{31} \rightarrow v_{31})} = \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 40$
$q_{base} = 10$

Ej. impactos grado 2 (b y d interaccionan)	
$\frac{\partial q(f_2=c)}{\partial f_1(a \rightarrow b)} = 50$	$\frac{\partial q(f_1=b)}{\partial f_2(c \rightarrow d)} = 90$
$\frac{\partial q(f_2=d)}{\partial f_1(a \rightarrow b)} = 60$	$\frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 40$
$\frac{\partial q(f_1=a)}{\partial f_2(c \rightarrow d)} = 80$	$q_{base} = 10$

Valor de q en los productos
$q_{prod_1} = q_{base} = 10$
$q_{prod_2} = q_{base} + \frac{\Delta q}{\Delta f_1(a \rightarrow b)} = 60$
$q_{prod_3} = q_{base} + \frac{\Delta q}{\Delta f_2(c \rightarrow d)} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 130$
$q_{prod_4} = q_{base} + \frac{\Delta q}{\Delta f_1(a \rightarrow b)} + \frac{\Delta q}{\Delta f_2(c \rightarrow d)} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 180$
$q_{prod_5} = q_{base} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 50$
$q_{prod_6} = q_{base} + \frac{\Delta q}{\Delta f_1(a \rightarrow b)} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 100$
$q_{prod_7} = q_{base} + \frac{\Delta q}{\Delta f_2(c \rightarrow d)} = 90$
$q_{prod_8} = q_{base} + \frac{\Delta q}{\Delta f_1(a \rightarrow b)} + \frac{\Delta q}{\Delta f_2(c \rightarrow d)} = 140$

Valor de q en los productos
$q_{prod_1} = q_{base} = 10$
$q_{prod_2} = q_{base} + \frac{\partial q(f_2=c)}{\partial f_1(a \rightarrow b)} = 60$
$q_{prod_3} = q_{base} + \frac{\partial q(f_1=a)}{\partial f_2(c \rightarrow d)} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 130$
$q_{prod_4} = q_{base} + \frac{\partial q(f_2=c)}{\partial f_1(a \rightarrow b)} + \frac{\partial q(f_1=b)}{\partial f_2(c \rightarrow d)} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 190$
$= q_{base} + \frac{\partial q(f_1=a)}{\partial f_2(c \rightarrow d)} + \frac{\partial q(f_2=d)}{\partial f_1(a \rightarrow b)} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 190$
$q_{prod_5} = q_{base} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 50$
$q_{prod_6} = q_{base} + \frac{\partial q(f_2=c)}{\partial f_1(a \rightarrow b)} + \frac{\Delta q}{\Delta f_3(f \rightarrow g)} = 100$
$q_{prod_7} = q_{base} + \frac{\partial q(f_1=a)}{\partial f_2(c \rightarrow d)} = 90$
$q_{prod_8} = q_{base} + \frac{\partial q(f_2=c)}{\partial f_1(a \rightarrow b)} + \frac{\partial q(f_1=b)}{\partial f_2(c \rightarrow d)} = 150$
$= q_{base} + \frac{\partial q(f_1=a)}{\partial f_2(c \rightarrow d)} + \frac{\partial q(f_2=d)}{\partial f_1(a \rightarrow b)} = 150$

F.3. Algoritmo de selección de productos

Para seleccionar los productos se ha desarrollado un algoritmo que selecciona el mínimo número de productos posible para detectar las interacciones existentes.

El algoritmo consiste en seleccionar aquellos productos con las combinaciones que permitan detectar las interacciones. Por lo tanto, es necesario seleccionar el grado de interacciones a detectar. Si el grado de interacción es 1 (es decir las características que impactan en la calidad no interaccionan entre ellas), por cada *feature* o característica f_i, se genera un producto con cada variante V_{ij} de esta característica, mientras que las demás características tienen una configuración de variantes igual. A partir del segundo feature, habrá un producto que coincida con alguno anteriormente generado. Con grado n (n características interaccionan), hay que seleccionar los productos de modo que todas las combinaciones de n características posibles sean consideradas.

Es aconsejable realizar una evaluación de productos incremental de modo que se comienza con los productos para detectar interacciones grado 2, y si tras analizar los resultados se detectan muchas interacciones se decide, evaluar también los productos necesarios para detectar interacciones grado 3 y así hasta detectar todas las interacciones.

El conjunto de productos seleccionados es el siguiente:

$$SPL = ProductosSeleccionados = \{SP_1, SP_2, ..., SP_m\} \tag{F.9}$$

- Si el grado es uno, tiene que existir algún producto seleccionado en el que cada una de las variantes de todas las características aparezca:

$$\exists SP_z \in SPL, \forall_{x \in NFF} \forall_{y \in NV_x} : V_{xy} \in NF_x, NF_x \in FL, V_{xy} \in SP_z$$

- Si el grado es dos, tiene que existir algún producto seleccionado en el que aparezca cada una de las combinaciones de las variantes dos a dos:

$$\begin{gathered} \forall_{x \in NFF} \forall_{y \in NV_x} : V_{xy} \in NF_x, NF_x \in FL \\ \forall_{j \in NFF} x \neq j, \forall_{k \in NV_j} : V_{jk} \in NF_j, NF_j \in FL \\ \exists SP_z \in SPL, V_{xy} \in SP_z, V_{jk} \in SP_z \end{gathered}$$

- Si el grado es n, tiene que existir algún producto seleccionado en el que aparezca cada una de las combinaciones de n variantes:

$$\begin{gathered} C = \{1, 2, 3, ..., n\} \forall_{i \in C} \forall_{x_i \in NFF} \forall_{y_i \in NV_{x_i}} : V_{x_i y_i} \in NF_{x_i}, NF_{x_i} \in FL \\ \exists SP_z \in SPL, V_{x_i y_i} \in SP_z \end{gathered}$$

Además de cumplir lo anteriormente especificado, el algoritmo busca que el número de productos sea el mínimo posible que permita cumplir las reglas mencionadas, para ello, hay que intentar que el máximo de combinaciones de

variantes coincida en el mismo producto, para que el número de productos sea mínimo.

Para eso, se comienza con la característica que más variantes tiene y se eligen los productos de modo que las combinaciones de n variantes se cubran. Así sucesivamente, con todas las características.

Siendo g: grado, f: feature y nf(f): NumeroVariantes de f.

```
int NumOpciones (feature f, grado g)
{ if (g==1) return (nv(f))
  else return (nv(f)*NumOpciones(f+1,g-1));}
```

El número de productos a seleccionar viene dado por la siguiente fórmula:

$$NumProductos = \sum_{f=1}^{NumFeatures-(g-1)} (NumOpciones(f,g) - NumOpciones(f-1,g-1) \tag{F.10}$$

Una vez que los productos han sido seleccionados, se instancian y se evalúan. Los resultados se comparan para detectar interacciones (ver siguiente sección).

F.4. Detectar interacciones

Si el impacto de cambiar de una variante a otra en una característica es diferente dependiendo del producto significa que esa característica interacciona con alguna otra característica. Por lo tanto, hay que obtener el impacto de cambiar de una variante concreta a otra variante para dos productos diferentes y comparar si los impactos son o no iguales. En caso de que sean iguales, las interacciones entre la característica cuya variante cambia y el resto de características que cambian de un producto a otro quedan descartadas.

Para detectar interacciones hay que comparar productos, en concreto hay que comparar el impacto por cambiar una variante (resultado de la resta del valor q de dos productos: P_a y P_b) contra el impacto por cambiar la misma variante (el resultado de la resta de q de otros dos productos: P_c y P_d), cumpliendo lo siguiente: P_a y P_b se diferencian sólo en una variante y P_c y P_d se diferencian

solamente en la misma variante. P_a y P_c (y P_b y P_d) se diferencian en una o más variantes.

$$\begin{gathered} a \in NP, b \in NP, c \in NP, d \in NP \\ \exists! x \in NFF, \exists! y \in NV_x, \exists! z \in NV_x : \\ V_{xy} \in P_a, V_{xy} \notin P_b, V_{xz} \notin P_a, V_{xz} \in P_b, \\ V_{xy} \in P_c, V_{xy} \notin P_d, V_{xz} \notin P_c, V_{xz} \in P_d \\ \exists! k \in NFF, k \neq x, \exists! l \in NV_k, \exists! m \in NV_k : \\ V_{kl} \in P_a, V_{kl} \in P_b, V_{km} \notin P_a, V_{km} \notin P_b, \\ V_{kl} \notin P_c, V_{kl} \notin P_d, V_{km} \in P_c, V_{km} \in P_d \end{gathered} \tag{F.11}$$

Todos los productos que cumplan lo anterior serán comparados. Una vez obtenidos los resultados hay que analizar qué significan.

Si

$$q_{Prod_a} - q_{Prod_b} = q_{Prod_c} - q_{Prod_d}$$

No se detecta ninguna interacción, por lo tanto se puede determinar que:

1. Si hay n características $(k_1, k_2 ... k_n)$ en las que las variantes cambian entre los dos pares de productos, se puede deducir lo siguiente (x,y,z,k,l,m están definidas en la ecuación F.11):

$$\forall i \in \{k_1, k_2, ..., k_n\} : \neg Int(f_{x(y \to z)}, f_{i(l \to m)}; q) \tag{F.12}$$

Si

$$q_{Prod_a} - q_{Prod_b} \neq q_{Prod_c} - q_{Prod_d}$$

Se detecta interacción. Para poder determinar qué características interaccionan hay que realizar un análisis más detallado que considera los siguientes casos:

1. Si sólo cambia una variante entre los dos pares de productos, se puede determinar la interacción:

$$\exists! k : Int(f_{x(y \to z)}, f_{i(l \to m)}; q) \tag{F.13}$$

2. Si hay n características $(k_1, k_2 ... k_n)$ en las que las variantes cambian entre los dos pares de productos, se puede deducir que al menos uno de las características interacciona con la característica x:

$$\exists i \in \{k_1, k_2, ..., k_n\} : Int(f_{x(y \to z)}, f_{i(l \to m)}; q) \tag{F.14}$$

   Hay que recordar que una interacción de grado dos no descarta interacciones de grados superiores.

Obtenemos interacciones descartadas, interacciones determinadas y posibles interacciones. Eliminando todas las interacciones descartadas de las listas de posibles interacciones nos podemos quedar con una lista de posibles interacciones que pueden ayudar a decidir el siguiente paso a realizar.

Además, un análisis de las posibles interacciones y de las características que más se repiten en las mismas puede ayudar a centrarse en las características que posiblemente son la causa de las interacciones.

En concreto, hay que intentar localizar las interacciones que siendo el número de interacciones reales, mínimamente cumplan que al menos haya una interacción para todas las listas de posibles interacciones detectadas.

A continuación una tabla con varios ejemplos. En el caso de la comparación Comp4 se pueden detectar las interacciones de la característica f2 con el resto de características. Los productos que se comparan, $Prod_1$ y $Prod_7$ sólo se diferencian en la variante de la característica f2; del mismo modo $Prod_8$ y $Prod_4$ se diferencian en la misma variante de la f2. A su vez, $Prod_1$ y $Prod_8$ (y $Prod_7$ y $Prod_4$) tienen diferentes variantes para las características f3 y f4. Por lo tanto, si el resultado de la comparación Comp4 es que son iguales, significa que f2 no interacciona ni con f3, ni con f4; incluso se puede deducir que f3 no interacciona con f4 ya que cambian ambos sin afectar a los impactos. Si por el contrario, el resultado de Comp4 fuera que las restas no son iguales (es decir, los impactos no son iguales) significa que existe una interacción de f2 con f3 o de f2 con f4 o de f3 con f4.

En el caso de que se detecte una posible interacción, para detallar entre qué características sucede suele ser necesario más información que se puede obtener del resto de comparaciones. Con todas las comparaciones se puede obtener la lista de posibles interacciones y a partir de esta lista se puede analizar la participación de cada una de las características, si hay alguna interacción detectable en todas las comparaciones, es muy probable que sea la causa. Por ejemplo, en el último ejemplo de la tabla (en la que todas las comparaciones de las restas son diferentes) todas las interacciones son posibles pero la interacción que se puede detectar en todas las comparaciones es la de f3 con f4 (sería suficiente esta interacción para que todas las comparaciones fueran diferentes). Partiendo de esta información, se seleccionan los productos que puedan ayudar a determinar las interacciones: descartando las interacciones menos probables.

Características	Productos evaluados	
$f_1 = \{v_{11}, v_{12}, v_{13}\} = \{a, b, c\}$	$Prod_1 = \{a, d, g, i\}$	$Prod_6 = \{c, f, h, j\}$
$f_2 = \{v_{21}, v_{22}\} = \{d, f\}$	$Prod_2 = \{b, d, g, i\}$	$Prod_7 = \{a, f, g, i\}$
$f_3 = \{v_{31}, v_{32}\} = \{g, h\}$	$Prod_3 = \{c, d, g, i\}$	$Prod_8 = \{a, d, h, j\}$
$f_4 = \{v_{41}, v_{42}\} = \{i, f\}$	$Prod_4 = \{a, f, h, j\}$	$Prod_9 = \{a, d, h, i\}$
	$Prod_5 = \{b, f, h, j\}$	$Prod_{10} = \{a, d, g, j\}$

Número	Comparación	Característica	Interacciones
Comp1	$q_{Prod_1} - q_{Prod_2} \Leftrightarrow q_{Prod_4} - q_{Prod_5}$	$f_1(a \rightarrow b)$	Todas
Comp2	$q_{Prod_2} - q_{Prod_3} \Leftrightarrow q_{Prod_5} - q_{Prod_6}$	$f_1(b \rightarrow c)$	Todas
Comp3	$q_{Prod_1} - q_{Prod_3} \Leftrightarrow q_{Prod_4} - q_{Prod_6}$	$f_1(a \rightarrow c)$	Todas
Comp4	$q_{Prod_1} - q_{Prod_7} \Leftrightarrow q_{Prod_8} - q_{Prod_4}$	$f_2(d \rightarrow f)$	$Int(f_2, f_3; q)$ $Int(f_2, f_4; q)$ $Int(f_3, f_4; q)$
Comp5	$q_{Prod_1} - q_{Prod_9} \Leftrightarrow q_{Prod_{10}} - q_{Prod_8}$	$f_3(g \rightarrow h)$	$Int(f_3, f_4; q)$
Comp6	$q_{Prod_1} - q_{Prod_{10}} \Leftrightarrow q_{Prod_9} - q_{Prod_8}$	$f_3(i \leftarrow j)$	$Int(f_3, f_4; q)$

Algunas posibilidades	Posibles interacciones de grado 2	Interacciones más probables
$Iguales(Comp1) \wedge Iguales(Comp2) \wedge Iguales(Comp3)$ $\wedge Iguales(Comp4) \wedge Iguales(Comp5) \wedge Iguales(Comp6)$	Ninguno. Impactos de grado 1	
$\neg Iguales(Comp1) \wedge \neg Iguales(Comp2) \wedge \neg Iguales(Comp3)$ $\wedge Iguales(Comp4) \wedge Iguales(Comp5) \wedge Iguales(Comp6)$	$Int(f_1, f_2; q) \vee$ $Int(f_1, f_3; q) \vee$ $Int(f_1, f_4; q)$	$Int(f_1, f_2; q) \vee$ $Int(f_1, f_3; q) \vee$ $Int(f_1, f_4; q)$
$\neg Iguales(Comp1) \wedge Iguales(Comp2) \wedge Iguales(Comp3)$ $\wedge Iguales(Comp4) \wedge Iguales(Comp5) \wedge Iguales(Comp6)$	$Int(f_{1(a\rightarrow b}, f_2; q) \vee$ $Int(f_{1(a\rightarrow b}, f_3; q) \vee$ $Int(f_{1(a\rightarrow b}, f_4; q)$	$Int(f_{1(a\rightarrow b}, f_2; q) \vee$ $Int(f_{1(a\rightarrow b}, f_3; q) \vee$ $Int(f_{1(a\rightarrow b}, f_4; q)$
$Iguales(Comp1) \wedge Iguales(Comp2) \wedge Iguales(Comp3)$ $\wedge \neg Iguales(Comp4) \wedge Iguales(Comp5) \wedge Iguales(Comp6)$	Imposible	
$\neg Iguales(Comp1) \wedge \neg Iguales(Comp2) \wedge \neg Iguales(Comp3)$ $\wedge \neg Iguales(Comp4) \wedge Iguales(Comp5) \wedge Iguales(Comp6)$	$Int(f_1, f_2; q) \vee$ $Int(f_1, f_3; q) \vee$ $Int(f_1, f_4; q) \vee$ $Int(f_2, f_3; q) \vee$ $Int(f_2, f_4; q)$	$Int(f_2, f_3; q) \vee$ $Int(f_2, f_4; q)$
$Iguales(Comp1) \wedge Iguales(Comp2) \wedge Iguales(Comp3)$ $\wedge Iguales(Comp4) \wedge \neg Iguales(Comp5) \wedge Iguales(Comp6)$	Imposible	
$\neg Iguales(Comp1) \wedge \neg Iguales(Comp2) \wedge \neg Iguales(Comp3)$ $\wedge \neg Iguales(Comp4) \wedge \neg Iguales(Comp5) \wedge \neg Iguales(Comp6)$	$Int(f_1, f_2; q) \vee$ $Int(f_1, f_3; q) \vee$ $Int(f_1, f_4; q) \vee$ $Int(f_2, f_3; q) \vee$ $Int(f_2, f_4; q) \vee$ $Int(f_3, f_4; q)$	$Int(f_3, f_4; q)$

Anexo G

Caso Línea de Productos Juegos Arcade

G.1. Introducción

La línea de productos pedagógica *The Arcade Game Maker* (AGM) desarrollado por SEI (*Software Engineering Institute*) [SEI 07a] es una línea de productos creada para facilitar el aprendizaje y la experimentación en las líneas de producto software.

La selección de esta línea viene motivada por dos razones, es una línea desarrollada para la experimentación, por lo que la documentación del diseño de la línea es pública y además la arquitectura de esta línea ya ha sido previamente evaluada utilizando ATAM [Clements 02] y también HoPLAA [Olumofin 05] posibilitando de esta manera comparar los resultados obtenidos.

El objetivo a la hora de aplicar el método en este caso de estudio ha sido validar la reducción de costes aplicando el método propuesto realizando una evaluación de la arquitectura software de la línea.

La línea de productos incluye tres juegos arcada diferentes (Brickle o ladrillos, Pong y bolos). Todos los productos tienen en común el uso de un conjunto de elementos gráficos o *sprites* (móviles y estacionarios), animación gráfica y un conjunto de reglas acerca de la interacción de las piezas del juego. Sin embargo, el tipo de elemento (disco, remo, bola de bolos, ladrillo...) y número es variante dependiendo del juego, así como las reglas, el comportamiento de los elementos y del entorno.

Los requisitos de calidad obligatorios son los siguientes:

- *Rendimiento*: La acción del juego tiene que ser lo suficientemente rápido para parecer continuo.
- *Usabilidad*: El juego tiene que ser atractivo para el jugador.

Para hacer el ejemplo más atractivo para la validación del método se ha añadido nueva variabilidad a la línea:

- Un nuevo juego: Se ha añadido el juego *SpaceWar*, un juego arcada bastante conocido (en la figura G.1 se puede observar la pantalla de un juego de este tipo). Los elementos móviles para el juego son naves espaciales (uno controlado por el jugador y el resto son naves enemigas) y balas disparadas por las naves.
- La posibilidad de seleccionar entre dos imágenes gráficas para los elementos. Estos iconos o imágenes para los elementos tienen tamaños diferentes con mayor o menor resolución.
- Sonido. El soporte de sonido es una funcionalidad opcional que proporciona una retroalimentación de sonido cada vez que hay una colisión de elementos.

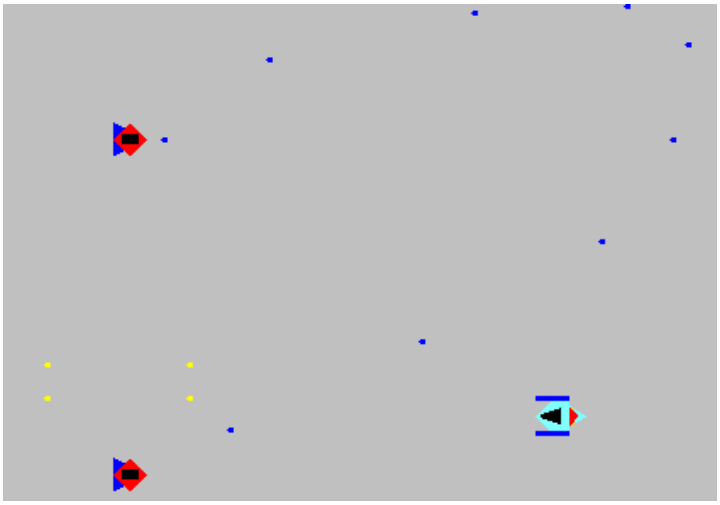


Figura G.1: Ejemplo de una pantalla del juego *SpaceWar*

G.2. Especificación de la Variabilidad

El primer paso es especificar las características funcionales, arquitecturales y de calidad así como los impactos cualitativos utilizando el modelo de características extendido.

En la figura G.2, se puede ver el modelo de características extendido y en la Tabla G.1 algunos de los impactos cualitativos (siendo "- -": impacta muy negativamente, "-": impacta negativamente, "+": impacta positivamente y "++': impacta muy positivamente). El juego *SpaceWar* tiene más elementos móviles que el resto de los juegos, por lo que el refresco es más complejo. En el mismo sentido, el juego de los bolos y el del los ladrillos tienen más elementos gráficos (*sprites*) que el juego *Pong*.

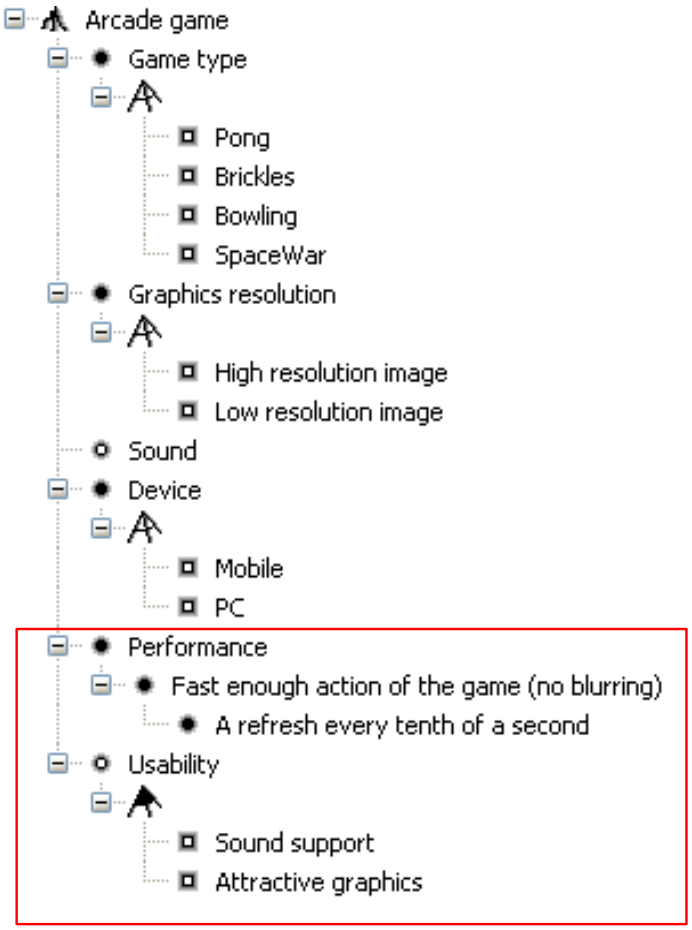


Figura G.2: Modelo CaFM de la LP Arcade

Tabla G.1: Impactos cualitativos

Feature	Impact	Quality feature
Brickles	-	A refresh every tenth of a second
Bowling	-	A refresh every tenth of a second
SpaceWar	- -	A refresh every tenth of a second
High resolution image	+	Attractive graphics
Sound	++	Sound Support
Mobile	- -	A refresh every tenth of a second

G.3. Validación de la calidad

En este caso de estudio, se han validado aspectos de rendimiento y usabilidad y se ha realizado una validación a nivel de diseño utilizando métodos de evaluación de arquitecturas software. Se ha utilizado el método PASA (*Performance assessment of software architectures*) [Williams 02] para evaluar aspectos de rendimiento. En el caso de la usabilidad se ha optado por no utilizar ningún método, aunque existan (como por ejemplo SALUTA [Folmer 03]) y realizar la cuantificación de los impactos de forma manual con ayuda de expertos y demostrar de este modo que la evaluación también puede estar basada en la participación de expertos.

Desarrollo del modelo genérico

Se ha generado el modelo de rendimiento genérico para el diagrama de secuencia del escenario más crítico (el refresco). Para que el juego le parezca continuo al usuario, es necesario realizar 10 refrescos en un segundo. Durante el refresco, hay que calcular la nueva localización de cada uno de los elementos móviles, chequear y manejar las colisiones entre elementos y pintar todos los elementos. El modelo de rendimiento genérico ha sido especificado utilizando un gráfico de ejecución [Smith 02] (ver figura G.3). En el gráfico de ejecución, cada uno de los pasos de procesamiento es especificado como un nodo con nodos de repetición para expresar cuántas veces se repite el nodo. Cada paso tiene una tabla con sus requisitos de recursos.

El gráfico de ejecución genérico tiene los mismos nodos o pasos de procesamiento en todos los productos. Sin embargo, hay aspectos que son variables:

- El número de repetición de los nodos de repetición: el número de elementos gráficos móviles y el número de elementos fijos así como el número de colisiones máximos dependerá del tipo de juego.

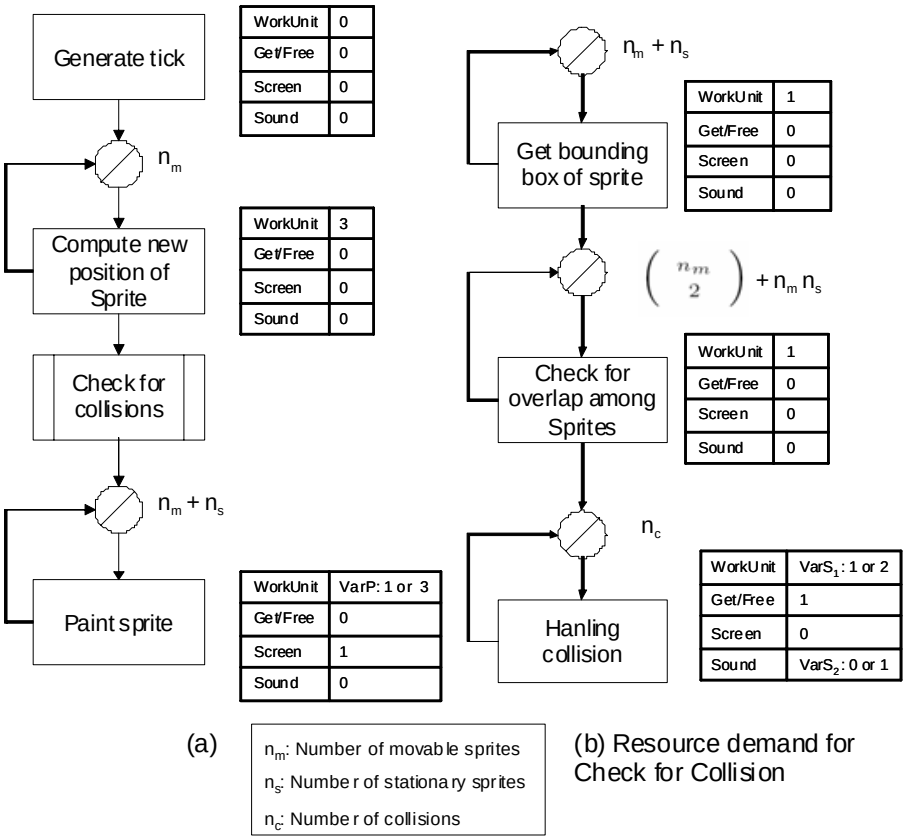


Figura G.3: El gráfico de ejecución para el escenario de refresco

- Calcular la posición de todos los elementos gráficos móviles (n_m)
- Buscar el límite de todos los elementos gráficos tanto móviles como estáticos o fijos ($n_m + n_s$)
- Chequear si hay solapamientos entre elementos para detectar colisiones. $\binom{n_m}{2} n_m n_s$. Mirar si los elementos móviles han colisionado entre ellos (la combinación de dos en dos de los elementos móviles) y también si los elementos móviles han colisionado contra los elementos fijos o estacionarios.
- Manejar las colisiones reproduciendo un sonido en caso de seleccionar esta opción (n_c).
- Pintar todos los elementos ($n_m + n_s$).

▪ Los requisitos de recursos software y en concreto el consumo de CPU también es variable en varios casos:
  - Al pintar, el consumo de CPU puede variar dependiendo de la resolución de las imágenes a pintar.
  - Al manejar una colisión, en caso de tener que reproducir un sonido se consume más CPU y hay un acceso a los altavoces.

La sobrecarga de procesamiento (ver tabla G.2) es una tabla de los requisitos de recursos del ordenador por cada demanda de recursos software.

Los recursos especificados en el ejemplo son los siguientes:

- *WorkUnit*: El número estimado de instrucciones de CPU para ejecutar una tarea simple. Las unidades de trabajo o *WorkUnits* son una forma conveniente de especificar el uso relativo de CPU de las tareas. En este ejemplo se utiliza un rango de 1 a 5; siendo 1 la tarea más simple y 5 la tarea más complicada. En la tabla se especifica el número para la tarea simple y el número de instrucciones para la tarea compleja será 5 veces la de la tarea simple.
- *Get/Free*: El número de llamadas a operaciones de gestión de memoria.
- *Screen*: El número de veces que se "dibuja" en la pantalla.
- *Sound*: El número de veces que se reproduce un sonido en los altavoces.

La sobrecarga de procesamiento (ver tabla G.2) ha sido estimada para dos dispositivos diferentes: un Nokia de la Serie60 con un CPU ARM-9 y un Pentium IV. Al introducir variabilidad en el dispositivo en el que va a desplegar el sistema, se introduce variabilidad en el tiempo de servicio: el tiempo de servicio para ejecutar 1 KInstrucciones ($VarS_1$), para dibujar en la pantalla ($VarS_2$) y

para reproducir un sonido en los altavoces ($VarS_3$) que depende del dispositivo. También hay otros aspectos que puede variar de un procesador a otro como por ejemplo el número de instrucciones de CPU necesarios para ejecutar una tarea pero esta variabilidad se ha obviado por facilitar la evaluación y porque se ha considerado suficiente introducir variabilidad en el tiempo de servicio.

Tabla G.2: Sobrecarga de procesamiento

Device	CPU	Display	Speaker
Quantity	1	1	1
Service Unit	Kinstr.	Units	Units

WorkUnit	20	0	0
Get/Free	2	0	0
Screen	1	1	0
Sound	1	0	1

Service Time	VarD1	VarD2	VarD3

Partiendo del gráfico de ejecución y de la tabla de sobrecarga de procesamiento se puede obtener la tabla con los recursos de ordenador totales por paso de procesamiento.

Tabla G.3: Recursos de ordenador totales por paso de procesamiento

Processing Step	**CPU (Kinstr)**	**Display**	**Sound**
Generate tick	0	0	0
Compute new position of Sprite	60	0	0
Get bounding box of sprite	20	0	0
Check for overlap among sprites	20	0	0
Handling collision	$20VarS_1 + 2 + VarS_2$	0	$VarS_2$
Paint sprite	20VarP + 1	1	0

Algunos de estos valores se deben multiplicar por el número de repetición del nodo de repetición en el que se encuentran los pasos de procesamiento.

Tabla G.4: Recursos de ordenador totales teniendo en cuenta las repeticiones

Processing Step	Times of Repetition	CPU (Kinstr)	Display	Sound
Generate tick	1	0	0	0
Compute new position of Sprite	n_m	60	0	0
Get bounding box of sprite	$n_m + n_s$	20	0	0
Check for overlap among sprites	$\binom{n_m}{2} n_m n_s$	20	0	0
Handling collision	n_c	$20VarS_1 + 2 + VarS_2$	0	$VarS_2$
Paint sprite	$n_m + n_s$	20VarP + 1	1	0
TOTAL	$60n_m + 20(n_m + n_s) + 20\left[\binom{n_m}{2} + n_m n_s\right] + (20VarS_1 + 2 + VarS_2)n_c + (20VarP + 1)(n_m + n_s)$		1	$VarS_2$

La suma de estos valores hay que multiplicarla por los tiempos de servicio. De esta forma se obtiene una fórmula genérica (ver ecuación G.1) para obtener el tiempo de refresco.

$$\left[60n_m + 20(n_m + n_s) + 20\left[\binom{n_m}{2} + n_m n_s\right] + (20VarS_1 + 2 + VarS_2)n_c + (20VarP + 1)(n_m + n_s)\right] VarD_1 + (n_m + n_s)VarD_2 + VarS_2 n_c VarD_3 \quad \text{(G.1)}$$

En este caso, la evaluación de los productos consiste en instanciar el modelo con los valores variables por cada producto y aplicar la fórmula con esos valores (ver tabla G.5).

Sin embargo, en muchos casos, la instanciación del modelo para los productos y el análisis del mismo, pueden ser muy costosos. A pesar de tener un modelo genérico es más rentable instanciar y evaluar sólo los productos seleccionados con el algoritmo. En este caso, aprovechando que ya tenemos una fórmula, vamos a aplicar la fórmula a todos los productos.

Tabla G.5: Matriz de trazabilidad de la variabilidad

Feature	Variante	nc	ns	nm	VarP	VarS1	VarS2	VarD1	VarD2	VarD3
Game type	Pong	1	4	3						
	Bowling	12	16	2						
	Blickles	3	36	4						
	SpaceWar	32	4	32						
Graphics Resolution	High				3					
	Low				1					
Sound	Yes					2	1			
	No					1	0			
Device	CPU							0,00001	0,002	0,002
	Mobile							0,00512	0,1	0,1

Con los resultados obtenidos es posible evaluar límites (el producto con el peor rendimiento) y detectar si se cumplen los escenarios obligatorios.

En este caso, no se cumplen, el peor producto respecto al rendimiento es el siguiente:

NumProducto	Características				Tiempo refresco
12	SpaceWar	HighResolution	Sound	Mobile	108,34304

No cumple el escenario seleccionado, así que se debe rediseñar la línea.

G.4. Derivación

Cuantificar

A partir de los resultados obtenidos se pueden cuantificar los impactos. Se

puede detectar que hay dos interacciones de tres *features*: Hay una interacción entre resolución, tipo juego y dispositivo y otra entre sonido, tipo juego y dispositivo.

Los impactos en el rendimiento se detallan en la siguiente tabla. Por ejemplo, el impacto en el tiempo de refresco es de +4,77359 milisegundos si *device* se cambia de *PC* a *mobile* (y el juego seleccionado es *Pong*, con *low resolution graphics* y sin soporte para sonido). Estos impactos se han logrado a partir de los valores de los productos obtenidos a partir de la fórmula G.1.

Tabla G.6: Impactos cuantitativos sobre tiempo de refresco en milisegundos

PC –>Mobile (Pong, Low, noSound)	4,77359
PC–>Mobile (Pong, Low, Sound)	4,9999
PC –>Mobile (Pong, High, Sound)	5,2443
noSound –>Sound (Pong, PC)	0,00221
noSound –>Sound (Pong, Mobile)	0,22852
Low –>High (Pong, PC)	0,0004
Low –>High (Pong, Mobile)	0,2448
PC–>Mobile (Pong, High, noSound)	5,01799
Pong–>Brickles (High, Sound, Mobile)	25,15868
Pong–>Brickles (Low, Sound, PC)	25,94458
Pong–>Brickles (Low, noSound, Mobile)	-0,75262
Pong–>Brickles (High, noSound, PC)	0,10097
Pong–>Bowling (High, Sound, Mobile)	5,66796
Pong–>Bowling (Low, Sound, PC)	7,82586
Pong–>Bowling (Low, noSound, Mobile)	-3,08407
Pong–>Bowling (High, noSound, PC)	0,02352
Pong–>SpaceWar (High, Sound, Mobile)	103,07544
Pong–>SpaceWar (Low, Sound, PC)	101,41134
Pong–>SpaceWar (Low, noSound, Mobile)	-1,03548
Pong–>SpaceWar (High, noSound, PC)	0,21011
...	

Respecto a la usabilidad, expertos en este atributo han cuantificado los impactos (ver tabla G.7) y se han definido pesos. *Sound support* tiene un peso de 30 y *attractive graphics* un peso de 70 para obtener usabilidad. Ejemplo: Un producto con la opción de sonido y ninguna opción más que ayude a la usabilidad tendría una usabilidad de 30 sobre 100 (ver los pesos en la figura G.4).

Niveles

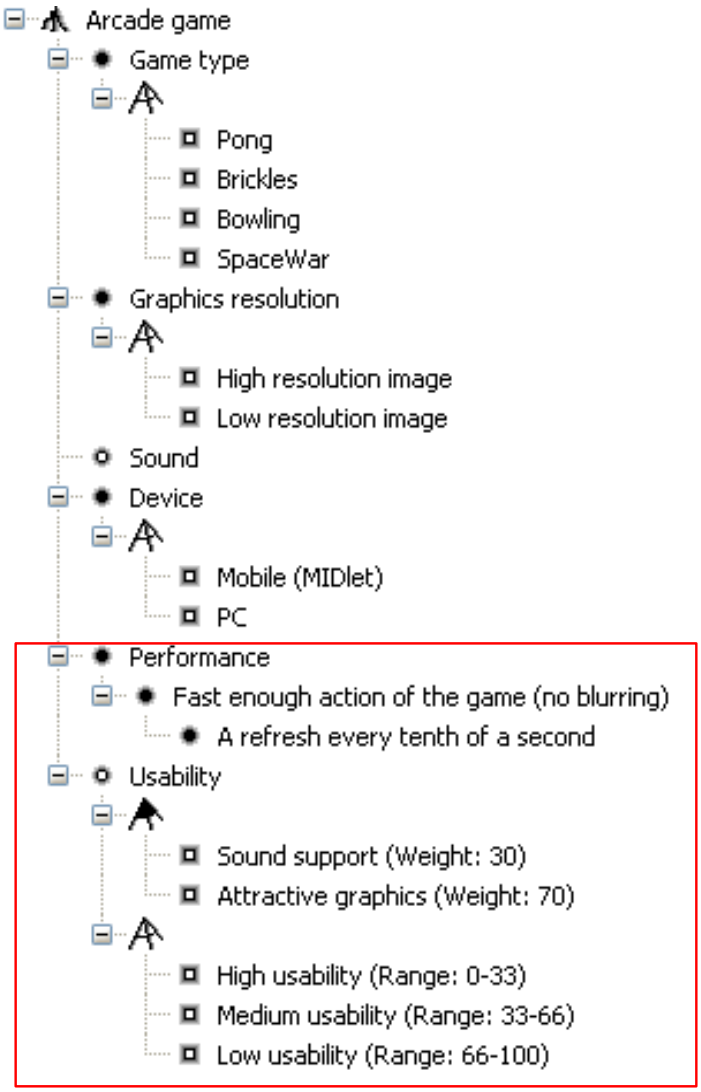


Figura G.4: Modelo CaFM de la LP Arcade con pesos y niveles

Tabla G.7: Impactos cuantitativos sobre usabilidad

Feature	**Impact**	**Quality feature**
High resolution image	80	Attractive graphics
Sound	100	Sound Support

Se han definido tres grupos o niveles de usabilidad para ayudar durante la derivación. El usuario puede seleccionar el grado o nivel de usabilidad deseado.

- Low usability [0-33)
- Medium usability [33-66)
- High usability [66-100]

Para rendimiento, no se han definido grupos o niveles porque se ha considerado que no es algo a tener en cuenta en la derivación siempre y cuando se cumpla el escenario obligatorio que marca el rendimiento mínimo aceptable.

Durante la derivación se puede ir observando el tiempo de refresco que se va a obtener (en este caso con que cumpla el mínimo requerido es suficiente pero en otros casos suele ser interesante ir viendo la evolución):

$Min + impactoindividual(condicional) + impactoindividual(condicional) + \rightarrow hastaobtenerelproductodeseado.$

Respecto a la usabilidad, al ir seleccionando las características también será posible ir viendo qué nivel de usabilidad se va logrando (*low*, *medium* o *high*).

Un ejemplo de derivación: Si se selecciona el juego *Brickles* sin sonido, con imágenes de alta resolución para PC. La usabilidad va a ser media (80 sobre 50 = 40) y el rendimiento se puede obtener a partir de los impactos de la tabla G.6:

0,02069 (el tiempo de refresco del producto con el menor tiempo = Pong, noSound, Low, PC)

+ 0,0004 Low −>High (Pong, PC)

+ 0,10097 Pong −>Brickles (High, noSound, PC)

= 0,12206

G.5. Conclusiones

Características evaluación				Resultados			
Atributos evaluados	Fase de evaluación	Estrategia de reducción	Técnicas de evaluación	Nº total productos	Nº productos evaluados	Reducción esfuerzo	Grado de confianza
Usabilidad Rendimiento	Diseño	Generación de un modelo genérico	Método de evaluación de arquitecturas software	32	Todos	Alto: el esfuerzo de especificar un modelo genérico es pequeño frente a evaluar todas los productos uno a uno	Muy alto: Todos los productos se evalúan por lo que se obtienen los mismos resultados que evaluando uno a uno

Obtener un modelo genérico de evaluación facilita mucho la evaluación de la línea. El esfuerzo de especificar el modelo genérico es pequeño comparado con el esfuerzo y coste de evaluar los productos uno a uno.

En este ejemplo en particular, el grado y número de interacciones es bastante alto debido a la naturaleza del atributo de calidad que se evalúa pero el modelo genérico permite evaluar todos los productos de forma sencilla y detectar las interacciones existentes.

Anexo H

Caso Línea de Productos Calculadora Software

H.1. Introducción

La línea de productos Calculadora software ha sido desarrollado específicamente para utilizarla durante la validación del método. La línea de productos se ha codificado utilizando AHEAD *Tool Suite* [Batory 04]. En AHEAD la derivación de los productos se puede realizar de forma automática de manera que obtener los productos de la línea es una tarea sencilla.

El objetivo a la hora de aplicar el método en este caso de estudio ha sido validar la reducción de costes aplicando el método propuesto, en concreto la estrategia de selección de un subconjunto de productos realizando una validación de los atributos mediante la ejecución y medición de los atributos de calidad.

La línea de productos incluye calculadoras para dispositivos con diferentes niveles de capacidad (Móviles, PCs...). Los dispositivos *wireless* o móviles imponen restricciones en ciertos atributos de calidad debido a su capacidad limitada respecto a los PCs. En concreto la línea de productos incluye productos que pueden desplegarse en los siguientes dispositivos:

- PC
- Móvil o *Pocket PC* que soporta configuración CDC (*Connected Device Configuration*) y con pantalla táctil

Las funcionalidades que incluye la línea se pueden observar en el modelo de características (ver H.2). Respecto a la calidad, los atributos de calidad más importantes en la línea son los siguientes:

- Usabilidad
- Rendimiento
- Eficiencia

Figura H.1: Ejemplo de calculadora en el emulador de Sony Ericsson 990

H.2. Especificación de la Variabilidad

También se han especificado los impactos cualitativos entre las características del modelo (siendo "- -": impacta muy negativamente, "-": impacta negativamente, "+": impacta positivamente y "++': impacta muy positivamente):

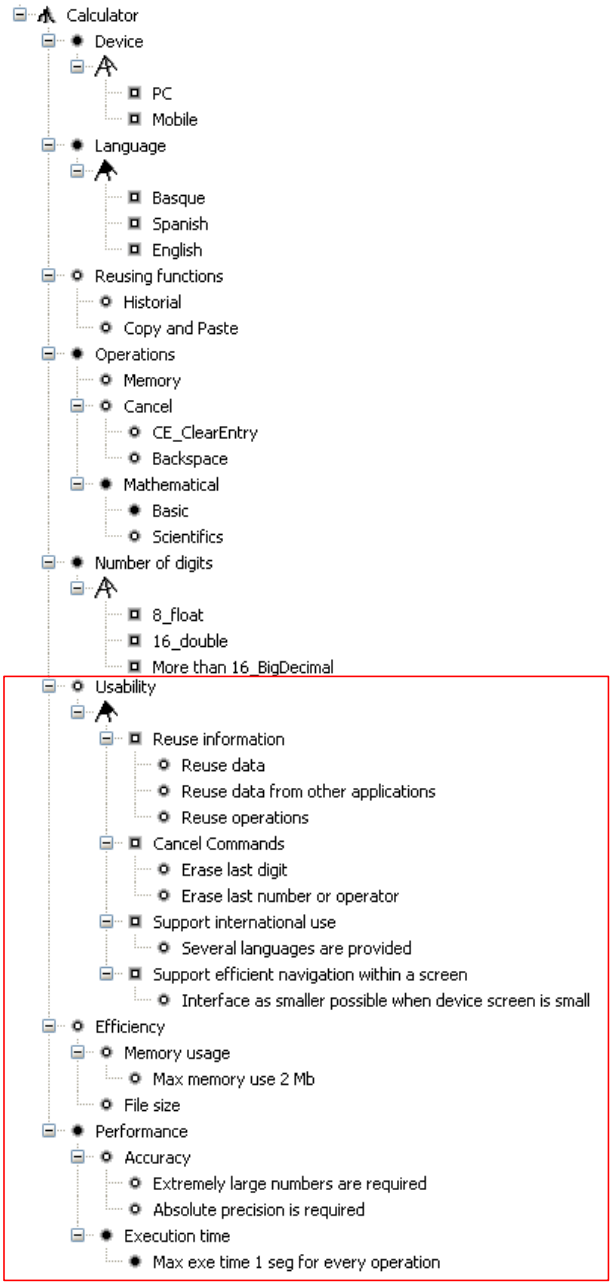


Figura H.2: Modelo CaFM de la LP calculadora

- Basque and Spanish and English impacts ++ on Several languages are provided
- (Basque and Spanish) or (Basque and English) or (Spanish and English) impacts + on Several languages are provided
- Basque and Spanish and English impacts ++ on File Size
- (Basque and Spanish) or (Basque and English) or (Spanish and English) impacts + on File Size
- Backspace impacts ++ on Erase last digit
- CEClearEntry impacts ++ on Erase last number or operator
- Memory impacts ++ on Reuse data
- Copy and past impacts ++ on Reuse data from other applications
- Historial impacts + on Reuse operations
- 16Float impacts + on Memory Usage
- More than 16BigDecimal impacts ++ on Memory Usage
- Historal and J2ME impacts - on Interface as smaller as possible when device is small
- Scientifics and J2ME impacts - on Interface as smaller as possible when decive is small
- All features impact + on File Size
- 16Float impacts + on Execution Time
- More than 16BigDecimal impacts ++ on Execution Time
- 16Float impacts + on Accuracy
- More than 16BigDecimal impacts ++ on Accuracy

H.3. Validación de la calidad

Se han seleccionado los aspectos de tiempo de ejecución y exactitud (*accuracy*) que corresponden al rendimiento y el espacio en disco y uso de memoria que corresponden a la eficiencia, así como la usabilidad para validarlos. La validación de los aspectos más medibles (tiempo de ejecución, espacio en disco y uso de memoria) se explica en primer lugar y posteriormente la validación de la usabilidad y la exactitud.

Tiempo de ejecución, espacio en disco y uso de memoria

Se va a medir el tiempo, el uso de memoria y el espacio en disco. Para medir el espacio en disco vamos a usar el comando ls de linux para listar archivos y en concreto ls -l **ls -l**: Lista los archivos con detalle, especificando para cada archivo sus permisos, el número de enlaces rígidos, el nombre del propietario, el grupo al que pertenece, el tamaño en *bytes*, y la fecha de modificación.

Se seleccionará el argumento de tamaño en *bytes*.

Pero para medir el tiempo y uso de memoria se han utilizado los siguientes métodos que java ofrece:

Para medir la memoria:	Para medir el tiempo:

Runtime runtime = Runtime.getRuntime();
long beforeMemory, afterMemory;
// Llamar al recolector de basura varias veces para asegurarse de que libere memoria
System.gc();System.gc();System.gc();
System.gc();System.gc();System.gc();
System.gc();System.gc();System.gc();
beforeMemory = runtime.totalMemory()-runtime.freeMemory();
// Realizar operación que se quiere medir
afterMemory = runtime.totalMemory()-runtime.freeMemory();
long size = afterMemory - beforeMemory;
	long last = 0, current = 0, tiempo = 0;
	last = System.currentTimeMillis( );
	// Realizar operación que se quiere medir
	current = System.currentTimeMillis( );
	tiempo = current - last ;
	System.out.println(" " + tiempo);
	Utilizando esto se consiguen muchos tiempos cercanos a cero o cero. Para más precisión se puede usar:
	long tiempo = System.nanoTime();

Antes de aplicar el algoritmo para seleccionar los productos:

Normalizar modelo de características

Transformar o normalizar el modelo de características en una lista plana de conjuntos de opciones o variantes que afectan a los aspectos de calidad seleccionados previamente.

En este caso, como todas las características afectan al tamaño, todas deben aparecer en la lista, para otros aspectos de calidad: tiempo de ejecución o uso de memoria no todos las características afectan, sin embargo como para el tamaño es necesario derivar todos los productos mencionados, se pueden utilizar los mismos para el resto de aspectos de calidad seleccionados. Así en algunos casos se obtendrán más datos, ya que están involucradas todas las características, y esto no es necesario para algunos de los aspectos de calidad.

Todas las transformaciones son automáticas menos la del lenguaje, se ha decidido que lo que puede implicar un impacto diferente en los aspectos de calidad es más el número de idiomas que la combinatoria de ciertos idiomas, por lo que se han creado tres variantes dependientes del número de idiomas.

Language = {1Lang, 2Lang, 3Lang}

NumberDigits={Float, Double, BigDecimal}

Device={PC,Mobile}

Memory={Memory, noMemory}

Ce={Ce, noCe}

Backspace={Backspace,noBackspace}

Copy&Paste = {C&P, noC&P}

Historial = {Historial, noHistorial}

Scientific = {Scientific, noScientific}

También se ha ordenado la lista de más a menos variantes.

Eliminar las dependencias

No es necesario debido a la ausencia de dependencias.

1. Obtener los productos a testear (*Aplicar el algoritmo de selección de productos*)

   Si se quieren detectar interacciones de grado 2, el número de productos a testear es de 24 productos de un total de 2688 productos posibles.

   Y los productos a evaluar serían los siguientes (F0 es la característida de language, F1: Number of digits, F3: Device, F4: Historial, F5: Copy and Paste, F6: CE, F7: Backspace y F8: Scientific).

Num	F0	F1	F2	F3	F4	F5	F6	F7	F8
0	1Lang	Float	PC	Historial	CyP	Memoria	ce	Backspace	Scientific
1	2Lang	Float	PC	Historial	CyP	Memoria	ce	Backspace	Scientific
2	3Lang	Float	PC	Historial	CyP	Memoria	ce	Backspace	Scientific
3	2Lang	Double	Mobile	noHistorial	noCyP	noMemoria	noCe	noBackspace	noScientific
4	1Lang	Double	Mobile	noHistorial	noCyP	noMemoria	noCe	noBackspace	noScientific
5	3Lang	Double	Mobile	noHistorial	noCyP	noMemoria	noCe	noBackspace	noScientific
6	3Lang	BigDecimal	PC	Historial	CyP	Memoria	ce	Backspace	Scientific
7	1Lang	BigDecimal	PC	Historial	CyP	Memoria	ce	Backspace	Scientific
8	2Lang	BigDecimal	PC	Historial	CyP	Memoria	ce	Backspace	Scientific
9	1Lang	Double	PC	Historial	CyP	Memoria	ce	Backspace	Scientific
10	1Lang	Float	Mobile	noHistorial	noCyP	noMemoria	noCe	noBackspace	noScientific
11	1Lang	BigDecimal	Mobile	noHistorial	noCyP	noMemoria	noCe	noBackspace	noScientific
12	1Lang	Float	Mobile	Historial	CyP	Memoria	ce	Backspace	Scientific
13	1Lang	Float	PC	noHistorial	noCyP	noMemoria	noCe	noBackspace	noScientific
14	1Lang	Float	PC	noHistorial	CyP	Memoria	ce	Backspace	Scientific
15	1Lang	Float	PC	Historial	noCyP	noMemoria	noCe	noBackspace	noScientific
16	1Lang	Float	PC	Historial	noCyP	Memoria	ce	Backspace	Scientific
17	1Lang	Float	PC	Historial	CyP	noMemoria	noCe	noBackspace	noScientific
18	1Lang	Float	PC	Historial	CyP	noMemoria	ce	Backspace	Scientific
19	1Lang	Float	PC	Historial	CyP	Memoria	noCe	noBackspace	noScientific
20	1Lang	Float	PC	Historial	CyP	Memoria	noCe	Backspace	Scientific
21	1Lang	Float	PC	Historial	CyP	Memoria	ce	noBackspace	noScientific
22	1Lang	Float	PC	Historial	CyP	Memoria	ce	noBackspace	Scientific
23	1Lang	Float	PC	Historial	CyP	Memoria	ce	Backspace	noScientific

2. Evaluar o testear los productos

En este caso, esta tarea ha consistido en derivar y ejecutar los productos con el código de medición necesario.

En el caso de los productos para móvil, los tiempos obtenidos están en milisegundos ya que no soporta el método NanoTime() y en el caso de los productos para PC, la medición se ha realizado en nanosegundos.

Los datos se han obtenido para dos tipos de operaciones diferentes:

- Una operación un poco complicada: "78467543267543218765 4.987654323467890246 * 67895.876543245678909876543 2345 / 5678.9876543245679876 + 4567890.98765432456789089765432135678987 65 * 7890765432"
- Una operación muy sencilla: "6 + 7"

No se quieren detectar las interacciones que causan variaciones poco representativas o insignificantes, por lo que se puede determinar el margen o porcentaje variante (en este caso 0,5 % de la media del atributo).

Tabla H.1: Resultados de las mediciones realizadas

Num	F0	F1	F2	F3	F4	F5	F6	F7	F8	FileSize	UsoMemOp1	UsoMemOp2	TiempOp1	TiempOp2
										Bytes			Nanoseg[1]	Nanoseg
0	1Lang	Float	PC	Historial	CyP	Memoria	Ce	Backspace	Scientific	51576	66176	33304	696295	37085
1	2Lang	Float	PC	Historial	CyP	Memoria	Ce	Backspace	Scientific	52713	66176	33304	729569	58354
2	3Lang	Float	PC	Historial	CyP	Memoria	Ce	Backspace	Scientific	53819	66176	33304	716643	28371
3	2Lang	Double	Mobile	noHistorial	noCyP	noMemoria	noce	noBackspace	noScientific	40483	37548	12972	0 (ms)	0 (ms)
4	1Lang	Double	Mobile	noHistorial	noCyP	noMemoria	noce	noBackspace	noScientific	40483	37548	12972	0 (ms)	0 (ms)
5	3Lang	Double	Mobile	noHistorial	noCyP	noMemoria	noce	noBackspace	noScientific	43388	37548	12972	0 (ms)	0 (ms)
6	3Lang	BigDecimal	PC	Historial	CyP	Memoria	Ce	Backspace	Scientific	63222	120528	62464	5645532	99537
7	1Lang	BigDecimal	PC	Historial	CyP	Memoria	Ce	Backspace	Scientific	60979	129912	62464	5711613	101258
8	2Lang	BigDecimal	PC	Historial	CyP	Memoria	Ce	Backspace	Scientific	62116	120528	62464	5689194	100391
9	1Lang	Double	PC	Historial	CyP	Memoria	Ce	Backspace	Scientific	51546	66176	33304	710016	37047
10	1Lang	Float	Mobile	noHistorial	noCyP	noMemoria	noce	noBackspace	noScientific	40473	37532	12956	0 (ms)	0 (ms)
11	1Lang	BigDecimal	Mobile	noHistorial	noCyP	noMemoria	noce	noBackspace	noScientific	40872	98580	38720	0 (ms)	0 (ms)
12	1Lang	Float	Mobile	Historial	CyP	Memoria	Ce	Backspace	Scientific	75786	38468	13892	0 (ms)	0 (ms)
13	1Lang	Float	PC	noHistorial	noCyP	noMemoria	noce	noBackspace	noScientific	28232	66648	33440	1256794	28303
14	1Lang	Float	PC	noHistorial	CyP	Memoria	Ce	Backspace	Scientific	44233	67240	34024	712335	28576
15	1Lang	Float	PC	Historial	noCyP	noMemoria	noce	noBackspace	noScientific	35554	66584	33376	1255108	27724
16	1Lang	Float	PC	Historial	noCyP	Memoria	Ce	Backspace	Scientific	48240	67240	34024	743639	28113
17	1Lang	Float	PC	Historial	CyP	noMemoria	noce	noBackspace	noScientific	38917	66584	33376	1233568	27936
18	1Lang	Float	PC	Historial	CyP	noMemoria	Ce	Backspace	Scientific	47674	67240	34024	728640	29037
19	1Lang	Float	PC	Historial	CyP	Memoria	noce	noBackspace	noScientific	42900	65688	32760	834359	28152
20	1Lang	Float	PC	Historial	CyP	Memoria	noce	Backspace	Scientific	50110	67240	34024	689569	28182
21	1Lang	Float	PC	Historial	CyP	Memoria	Ce	noBackspace	noScientific	44460	65576	32760	836364	27979
22	1Lang	Float	PC	Historial	CyP	Memoria	Ce	noBackspace	Scientific	50733	66208	33312	695024	28543
23	1Lang	Float	PC	Historial	CyP	Memoria	Ce	Backspace	noScientific	45310	65944	33040	710277	28119

Tabla H.2: Comparación de resultados y posibles interacciones

NumFea	Var1	Var2	Productos1	Productos2	Resta1	Resta2	FileSize	Resta1	Resta2	UsoMemOp1	Resta1	Resta2	UsoMemOp2
0	1Lang	2Lang	0 vs 1	4 vs 3	1137.0	0.0	Interacción	0.0	0.0	Iguales	0.0	0.0	Iguales
	0	0	0 vs 1	7 vs 8	1137.0	1137.0	Iguales	0.0	-9384.0	Interacción	0.0	0.0	Iguales
	0	0	4 vs 3	7 vs 8	0.0	1137.0	Interacción	0.0	-9384.0	Interacción	0.0	0.0	Iguales
0	1Lang	3Lang	0 vs 2	4 vs 5	2243.0	2905.0	Interacción	0.0	0.0	Iguales	0.0	0.0	Iguales
	0	0	0 vs 2	7 vs 6	2243.0	2243.0	Iguales	0.0	-9384.0	Interacción	0.0	0.0	Iguales
	0	0	4 vs 5	7 vs 6	2905.0	2243.0	Interacción	0.0	-9384.0	Interacción	0.0	0.0	Iguales
1	Float	BigDecimal	0 vs 7	1 vs 8	9403.0	9403.0	Iguales	63736.0	54352.0	Interacción	29160.0	29160.0	Iguales
	0	0	0 vs 7	2 vs 6	9403.0	9403.0	Iguales	63736.0	54352.0	Interacción	29160.0	29160.0	Iguales
	0	0	0 vs 7	10 vs 11	9403.0	399.0	Interacción	63736.0	61048.0	Casi Iguales	29160.0	25764.0	Interacción
	0	0	1 vs 8	2 vs 6	9403.0	9403.0	Iguales	54352.0	54352.0	Iguales	29160.0	29160.0	Iguales
	0	0	1 vs 8	10 vs 11	9403.0	399.0	Interacción	54352.0	61048.0	Interacción	29160.0	25764.0	Interacción
	0	0	2 vs 6	10 vs 11	9403.0	399.0	Interacción	54352.0	61048.0	Interacción	29160.0	25764.0	Interacción
1	Float	Double	0 vs 9	10 vs 4	-30.0	10.0	Interacción	0.0	16.0	Interacción	0.0	16.0	Interacción
2	PC	Mobile	0 vs 12	13 vs 10	24210.0	12241.0	Interacción	-27708.0	-29116.0	Casi Iguales	-19412.0	-20484.0	Casi Iguales
3	historial8	nohistorial	0 vs 14	15 vs 13	-7343.0	-7322.0	Casi Iguales	1064.0	64.0	Interacción	720.0	64.0	Interacción

[1]salvo que se indique lo contrario

4	copypaste	nocopypaste	0 vs 16	17 vs 15	-3336.0	-3363.0	Casi Iguales	1064.0	0.0	Interacción	720.0	0.0	Interacción
5	memoria	nomemoria	0 vs 18	19 vs 17	-3902.0	-3983.0	Casi Iguales	1064.0	896.0	Interacción	720.0	616.0	Interacción
6	Ce	noce	0 vs 20	21 vs 19	-1466.0	-1560.0	Casi Iguales	1064.0	112.0	Interacción	720.0	0.0	Interacción
7	backspace	nobackspace	0 vs 22	23 vs 21	-843.0	-850.0	Casi Iguales	32.0	-368.0	Interacción	8.0	-280.0	Interacción
8	scientific	noscientific	0 vs 23	22 vs 21	-6266.0	-6273.0	Casi Iguales	-232.0	-632.0	Interacción	-264.0	-552.0	Interacción
0	2Lang	3Lang	1 vs 2	3 vs 5	1106.0	2905.0	Interacción	0.0	0.0	Iguales	0.0	0.0	Iguales
	0	0	1 vs 2	8 vs 6	1106.0	1106.0	Iguales	0.0	0.0	Iguales	0.0	0.0	Iguales
	0	0	3 vs 5	8 vs 6	2905.0	1106.0	Interacción	0.0	0.0	Iguales	0.0	0.0	Iguales
1	Double	BigDecimal	4 vs 11	9 vs 7	389.0	9433.0	Interacción	61032.0	63736.0	Casi Iguales	25748.0	29160.0	Interacción

3. Comparar los resultados y detectar interacciones

   El siguiente paso es comparar los resultados obtenidos.

   Tamaño de fichero

   Una vez detectadas las interacciones, hay que descubrir qué características interaccionan. Para ello, hay que analizar qué características cambian en los productos que interaccionan, cuáles se mantienen y también qué características cambian o se mantienen en los productos que no interaccionan. Analizando los resultados: Existen interacciones que se pueden descartar (características que cambian de un par de productos al otro y no hay interacción):

   - Feature 0 y Feature 1, Feature 3 y Feature 4, Feature 3 y Feature 5, Feature 3 y Feature 6, Feature 3 y Feature 7, Feature 3 y Feature 8, Feature 4 y Feature 5, Feature 4 y Feature 6, Feature 4 y Feature 7, Feature 4 y Feature 8, Feature 5 y Feature 6, Feature 5 y Feature 7, Feature 5 y Feature 8, Feature 6 y Feature 7, Feature 6 y Feature 8, Feature 7 y Feature 8

   Posibles interacciones

   - Feature 0 y Feature 2, Feature 0 y Feature 3, Feature 0 y Feature 4, Feature 0 y Feature 5, Feature 0 y Feature 6, Feature 0 y Feature 7, Feature 0 y Feature 8, Feature 1 y Feature 2, Feature 1 y Feature 3, Feature 1 y Feature 4, Feature 1 y Feature 5, Feature 1 y Feature 6, Feature 1 y Feature 7, Feature 1 y Feature 8, Feature 2 y Feature 2, Feature 2 y Feature 3, Feature 2 y Feature 4, Feature 2 y Feature 5, Feature 2 y Feature 6, Feature 2 y Feature 7, Feature 2 y Feature 8

   Analizando las características que cambian en las interacciones, hay una que está presente en todas las interacciones detectadas (Feature 2). En este caso, parece que *device* (PC, *mobile*) interacciona con varias características por lo que causa la mayoría de las interacciones.

   Para comprobar que en realidad todas las interacciones son debidas a *device*, es más sencillo medir productos orientados a descartar el resto de características que también cambian en las interacciones.

   A pesar de cambiar cuando existe una interacción, si además no cambia cuando no hay interacción, siendo los mismos productos salvo por una característica, se demuestra que esa característica no es la causa de la interacción.

   Para comprobar que el feature 1 y el feature 0 no interaccionan salvo con *device*, hay que evaluar otros 18 productos:

Num	F0	F1	F2	F3	F4	F5	F6	F7	F8
24	1Lang	Double	PC	noHistorial	CyP	Memoria	ce	Backspace	Scientific
25	1Lang	Double	PC	Historial	noCyP	Memoria	ce	Backspace	Scientific
26	1Lang	Double	PC	Historial	CyP	noMemoria	ce	Backspace	Scientific
27	1Lang	Double	PC	Historial	CyP	Memoria	noCe	Backspace	Scientific
28	1Lang	Double	PC	Historial	CyP	Memoria	ce	noBackspace	Scientific
29	1Lang	Double	PC	Historial	CyP	Memoria	ce	Backspace	noScientific
30	2Lang	Float	PC	noHistorial	CyP	Memoria	ce	Backspace	Scientific
31	2Lang	Float	PC	Historial	noCyP	Memoria	ce	Backspace	Scientific
32	2Lang	Float	PC	Historial	CyP	noMemoria	ce	Backspace	Scientific
33	2Lang	Float	PC	Historial	CyP	Memoria	noCe	Backspace	Scientific
34	2Lang	Float	PC	Historial	CyP	Memoria	ce	noBackspace	Scientific
35	2Lang	Float	PC	Historial	CyP	Memoria	ce	Backspace	noScientific
36	1Lang	BigDecimal	PC	noHistorial	CyP	Memoria	ce	Backspace	Scientific
37	1Lang	BigDecimal	PC	Historial	noCyP	Memoria	ce	Backspace	Scientific
38	1Lang	BigDecimal	PC	Historial	CyP	noMemoria	ce	Backspace	Scientific
39	1Lang	BigDecimal	PC	Historial	CyP	Memoria	noCe	Backspace	Scientific
40	1Lang	BigDecimal	PC	Historial	CyP	Memoria	ce	noBackspace	Scientific
41	1Lang	BigDecimal	PC	Historial	CyP	Memoria	ce	Backspace	noScientific

Tras comparar los resultados, hay una pequeña interacción entre Feature 0 y Feature 4 y una interacción considerable entre Feature 1 y Feature 8, cuando el tipo de dato es *BigDecimal*. Para introducir la característica de *Scientific* es necesario incluir una clase más y esto tiene repercusión en el tamaño. Sin olvidar, la interacción que es debida al tipo de dispositivo que interacciona con el resto de características.

Teniendo en cuenta las interacciones detectadas, para poder cuantificar los impactos, es necesario testear más productos:

Num	F0	F1	F2	F3	F4	F5	F6	F7	F8
42	1Lang	BigDecimal	Mobile	Historial	CyP	Memoria	ce	Backspace	Scientific
43	2Lang	Float	Mobile	Historial	CyP	Memoria	ce	Backspace	Scientific
44	1Lang	Float	Mobile	noHistorial	CyP	Memoria	ce	Backspace	Scientific
45	1Lang	Float	Mobile	Historial	noCyP	Memoria	ce	Backspace	Scientific
46	1Lang	Float	Mobile	Historial	CyP	noMemoria	ce	Backspace	Scientific
47	1Lang	Float	Mobile	Historial	CyP	Memoria	noCe	Backspace	Scientific
48	1Lang	Float	Mobile	Historial	CyP	Memoria	ce	noBackspace	Scientific
49	1Lang	Float	Mobile	Historial	CyP	Memoria	ce	Backspace	noScientific

En total se evalúan 50 productos de 2688 (Evaluando el 1,86 % de los productos es suficiente para detectar y cuantificar los impactos detectados)

Tiempo de ejecución

El tiempo de ejecución es inferior al milisegundo en todos los productos y además hay muchas interacciones. Realmente no se desea hacer niveles respecto al tiempo de ejecución, lo que se deseaba era poder asegurar que el escenario obligatorio de respuesta en menos de un 1 segundo se cumple. Viendo los resultados se puede asegurar que se cumple sin ningún problema para todos los productos, por lo que no se van a evaluar más productos para especificar las interacciones y los impactos.

Uso de memoria

Respecto al uso de memoria sucede algo similar que con el tiempo de ejecución, se desea comprobar que el uso de memoria no es superior a 2Mb en los casos que se despliega en un móvil. No es necesario detectar todas las interacciones y cuantificar todos los impactos al menos que se quieran definir niveles porque

en derivación el nivel de uso de memoria sea importante.

Para ver si el escenario se cumple hay que evaluar el producto que según los impactos se considera que va a tener el uso de memoria más alto:

Num	F0	F1	F2	F3	F4	F5	F6	F7	F8
44	3Lang	BigDecimal	Mobile	Historial	CyP	Memoria	ce	Backspace	Scientific

El resultado es inferior a 2Mb por mucho por lo que este escenario se cumple.

H.4. Derivación

Y partiendo de los datos, los impactos cuantitativos para tamaño de fichero son los siguientes:

Características	Impacto
$1Lang \rightarrow 2Lang(PC)$	1137.0
$2Lang \rightarrow 3Lang(PC)$	1106.0
$1Lang \rightarrow 3Lang(PC)$	2243.0
$Float \rightarrow Double(PC)$	-30.0
$Double \rightarrow BigDecimal(PC, noScientific)$	246
$Float \rightarrow BigDecimal(PC, noScientific)$	246
$Double \rightarrow BigDecimal(PC, Scientific)$	9433.0
$Float \rightarrow BigDecimal(PC, Scientific)$	9403.0
$1Lang \rightarrow 2Lang(Mobile)$	1459
$2Lang \rightarrow 3Lang(Mobile)$	1446
$1Lang \rightarrow 3Lang(Mobile)$	2905.0
$Float \rightarrow Double(Mobile)$	10.0
$Double \rightarrow BigDecimal(Mobile, noScientific)$	389.0
$Float \rightarrow BigDecimal(Mobile, noScientific)$	399.0
$Double \rightarrow BigDecimal(Mobile, Scientific)$	10210
$Float \rightarrow BigDecimal(Mobile, Scientific)$	10210
$PC \rightarrow Mobile(1Lang, Float, noHistorial, nocyp, nomemoria, noce, nobackspace, noscientific)$	12241
$noHistorial \rightarrow Historial(PC)$	7332,5
$noCyP \rightarrow CyP(PC)$	3349,5
$nomemoria \rightarrow memoria(PC)$	3942,5
$noce \rightarrow ce(PC)$	1513
$nobackspace \rightarrow backspace(PC)$	846,5
$noscientific \rightarrow scientific(PC, FloatorDouble)$	6269,5
$noscientific \rightarrow scientific(PC, BigDecimal)$	15423
$noHistorial \rightarrow Historial(Mobile)$	9340
$noCyP \rightarrow CyP(Mobile)$	4180
$nomemoria \rightarrow memoria(Mobile)$	5104
$noce \rightarrow ce(Mobile)$	1935
$nobackspace \rightarrow backspace(Mobile)$	1084
$noscientific \rightarrow scientific(Mobile, FloatorDouble)$	8025
$noscientific \rightarrow scientific(Mobile, BigDecimal)$	16423

También se han definido niveles para el tamaño de fichero:

- Very Low [28202-37718,8)
- Low [37718,8 -47235,6)
- Medium [47235,6- 56752,4)
- High [56752,4-66269,2)
- Very High [66269,2-75786]

Usabilidad

La usabilidad no se ha medido o evaluado siguiendo un método existente. Se ha utilizado un enfoque más intuitivo y subjetivo para medir o valorar la usabilidad. Por cada escenario y *concern* se han especificado pesos (como ayuda a lograr la característica padre) y a nivel de usabilidad se han realizado grupos (ver figura H.3 para observar niveles y pesos).

A partir de los impactos cualitativos:

- Basque and Spanish and English impacts ++ on Several languages are provided
- (Basque and Spanish) or (Basque and English) or (Spanish and English) impacts + on Several languages are provided
- Backspace impacts ++ on Erase last digit
- CEClearEntry impacts ++ on Erase last number or operator
- Memory impacts ++ on Reuse data
- Copy and past impacts ++ on Reuse data from other applications
- Historial impacts + on Reuse operations
- Historal and J2ME impacts - on Interface as smaller as possible when decive is small
- Scientifics and J2ME impacts - on Interface as smaller as possible when decive is small

Se cuantifican los impactos cualitativos $(- \rightarrow -50, + \rightarrow 50, ++ \rightarrow 100)$

Impactos cuantitativos:

- Basque and Spanish and English impacts (100) on Several languages are provided
- (Basque and Spanish) or (Basque and English) or (Spanish and English) impacts (50) on Several languages are provided
- Backspace impacts (100) on Erase last digit

Tabla H.3: Rangos para 3 grupos o niveles en usabilidad

Nivel de usabilidad	Rango
Bajo	[0-33]
Medio	(33-66]
Alto	(66-100]

- CEClearEntry impacts (100) on Erase last number or operator
- Memory impacts (100) on Reuse data
- Copy and past impacts (100) on Reuse data from other applications
- Historial impacts (50) on Reuse operations
- Historal and J2ME impacts (-50) on Interface as smaller as possible when decive is small
- Scientifics and J2ME impacts (-50) on Interface as smaller as possible when decive is small

Impactos teniendo en cuenta pesos:

- Basque and Spanish and English impacts (20) on Usability
- (Basque and Spanish) or (Basque and English) or (Spanish and English) impacts (10) on Usability
- Backspace impacts (100 de 30 de 30 = 9) on Usability
- CEClearEntry impacts (100 de 70 de 30 =21) on Usability

Accuracy

Respecto a la exactitud, también se ha realizado la evaluación manualmente.

A partir de los impactos cualitativos:

- 16Float impacts + on Accuracy
- More than 16BigDecimal impacts ++ on Accuracy

Se cuantifican los impactos cualitativos ($+ \rightarrow 50, ++ \rightarrow 100$)

Impactos cuantitativos:

- 16Float impacts (50) on Accuracy
- More than 16BigDecimal impacts (100) on Accuracy

Tabla H.4: Rangos para 3 grupos o niveles en *accuracy*

Nivel de exactitud	Rango
Bajo	[0-33]
Medio	(33-66]
Alto	(66-100]

Y se definen unas dependencias entre los escenarios opcionales definidos y el tipo de datos:

- "Extremely large number are required" requires More than 16BigDecimal
- "Absolute precision is required" requires More than 16BigDecimal

Con los niveles y pesos definidos y los impactos cuantificados, la derivación se facilita mucho.

Se puede ir viendo el nivel de usabilidad, exactitud o el nivel del tamaño de fichero que va a tener la aplicación de un modo muy sencillo: se parte del tamaño del producto mínimo, se van sumando los impactos y se observa en qué nivel se encuentra.

H.5. Enfoque *bottom-up*

Se han evaluado los 2688 productos y comparado los resultados. Todos los productos cumplen los escenarios de uso de memoria y tiempo, tal como se había observado con el método.

Acerca del tamaño de fichero, el método clasifica correctamente (en el nivel correspondiente) el 98 % de los productos.

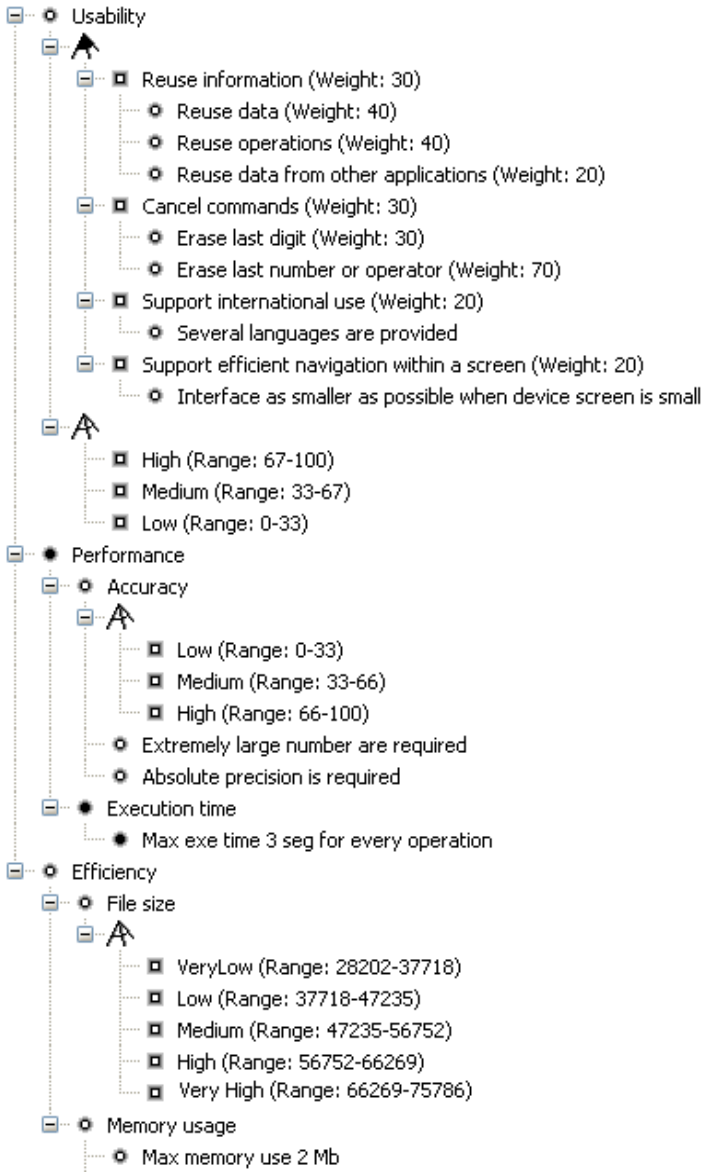


Figura H.3: Características de calidad del modelo CaFM de la calculadora para derivación

H.6. Conclusiones

Características evaluación				Resultados			
Atributos evaluados	Fase de evaluación	Estrategia de reducción	Técnicas de evaluación	Nº total productos	Nº productos evaluados	Reducción esfuerzo	Grado de confianza
Usabilidad Rendimiento Eficiencia	Implementación	Selección de un subconjunto de productos	Medición de los aspectos de calidad mediante ejecución de los productos	2688	44	98 %	98 %

El porcentaje de los productos a evaluar es muy pequeño para evaluar toda la línea, lo que demuestra que aplicando el método se reduce considerablemente el esfuerzo y coste de validación.

También se ha aplicado un enfoque *bottom up* y se han evaluado todos los productos y determinado los impactos. Los resultados obtenidos aplicando el método y evaluando todos los productos son muy similares.

Anexo I

Caso GPL (Graph Product line)

I.1. Introducción

La línea de productos grafo es una línea de productos que fue publicada como un problema estándar para evaluar metodologías de línea de productos [Lopez-Herrejon 01] [Batory 05]. Se ha trabajado con esta línea de productos implementada con AHEAD.

El objetivo a la hora de aplicar el método en este caso de estudio ha sido validar la reducción de costes aplicando el método propuesto, en concreto la estrategia de selección de un subconjunto de productos realizando una validación de los atributos mediante la ejecución y medición de los atributos de calidad, en este caso no sólo atributos de calidad operacionales sino también de desarrollo.

Esta línea de productos genera aplicaciones para trabajar con grafos (ver figura I.1), las aplicaciones pueden trabajar con uno o más algoritmos de grafos. El tipo de grafo con los que pueden trabajar también varia: puede ser con pesos o sin pesos y dirigidos o no dirigidos. Además, un producto puede tener como máximo un algoritmo de búsqueda que puede ser de dos tipos: *depth-first search* (DFS) or *breadth-first search* (BFS). Algunos algoritmos requieren trabajar con grafos de ciertas características (con pesos, no dirigido...). Estas restricciones se representan como dependencias. La implementación del producto también es variante: existen tres diseños diferentes dependiendo de como se representa la información del grafo.

Los algoritmos:

- *Vertex Numbering (Number)*: Este algoritmo numera todos los nodos en un grafo.
- *Connected Graphs (Connected)*: Testea si un grafo es conectado. En un grafo no dirigido, un grafo es conectado si todos los vértices están conectados dos a dos. Dos nodos están conectados si y sólo si existe un camino entre ellos.
- *Strongly Connected Graphs (StronglyConnected)*: Chequea si un grafo está *strongly connected*: En un grafo dirigido, si por cada par de vértices u y v existe un camino desde u a v y un camino desde v a u.
- *Cycle Checking (Cycle)*: Determina si hay ciclos en un grafo. Un ciclo en grafos dirigidos debe tener al menos dos aristas, mientras que en grafos no dirigidos tiene que tener al menos 3 aristas.
- *Prim's Minimum Spanning Tree (MSTPrim)*: El algoritmo Prim es un algoritmo para buscar el spanning tree mínimo para un grafo conectado y con pesos. Esto significa que busca un subconjunto de aristas que forman un árbol que incluyen todos los vértices, y donde el total de pesos de todas las aristas en el árbol está minimizado.
- *Kruskal's Minimum Spanning Tree (MSTKruskal)*: El algoritmo de Kruskal es otro algoritmo para buscar el *spanning tree* mínimo para un grafo conectado y con pesos.

Algoritmos de búsqueda (Src)

- *Breadth First Search (BFS)*: es un algoritmo de búsqueda que comienza en el nodo raíz y explora todos los nodos vecinos. Luego, por cada uno de esos nodos más cercanos, explora sus nodos vecinos no explorados, y así hasta encontrar su objetivo.
- *Depth First Search* (DFS): es un algoritmo de búsqueda que realiza la exploración en profundidad. Intuitivamente, el algoritmo empieza en la raíz y explora lo máximo posible a lo largo de cada rama antes de volver hacia atrás o desandar.

Y las dependencias

- Number implies Gtp and Src
- Connected implies Undirected and Src
- StrongC implies Directed and DFS
- Cycle implies Gtp and DFS
- MSTKruskal or MSTPrim implies Undirected and Weighted
- MSTKruskal or MSTPrim implies not (MSTKruskal and MSTPrim)

Respecto a los atributos de calidad, uno de los aspectos más importantes puede resultar el rendimiento, la respuesta en tiempo cuando se trabaja con grafos muy grandes.

A nivel de diseño, la forma de representar la información de los grafos va a impactar en los atributos de calidad. Hay tres diseños posibles:

- Solo vértices (*OnlyVertices*): Solamente se guarda información de los vértices y listas de adyacencia. Las aristas son implícitas: su existencia se puede inferir desde una lista da adyacencia.
- Con Vecinos (*WithNeighbours*): Se guarda información de los vértices y una lista de objetos vecinos. De este modo las anotaciones de las aristas (pesos...) se pueden codificar como refinamientos (campos extras) de la clase vecino.
- Con aristas (*WithEdges*): Se guarda información explícita de las aristas además de los vértices y los vecinos.

El algoritmo MSTKruskal requiere trabajar con aristas y trabajar con los otros diseños complica mucho la implementación, por lo tanto hay una dependencia, y la selección de este algoritmo implica un diseño concreto.

- Kruskal requires WithEdges

Sin embargo, en el caso de los demás algoritmos el tipo de diseño no está restringido y se sabe que puede influir en ciertos atributos de calidad.

Los atributos de calidad que se han evaluado han sido el tiempo de respuesta y la mantenibilidad.

I.2. Especificación de la variabilidad

La figura I.1 muestra el modelo de características extendido del ejemplo. Los atributos de calidad han sido especificados y caracterizados. Por ejemplo, *Performance* entendido como tiempo de ejecución y con un escenario obligatorio: "*Execution time<1sec with graphs of 100k edges*". Este escenario solamente tiene que ser chequeado cuando el rendimiento es importante para un producto, ya que el rendimiento es opcional. Y la mantenibilidad caracterizada utilizando una métrica (*maintainability index*), de esta forma es posible medir la mantenibilidad de una forma no subjetiva.

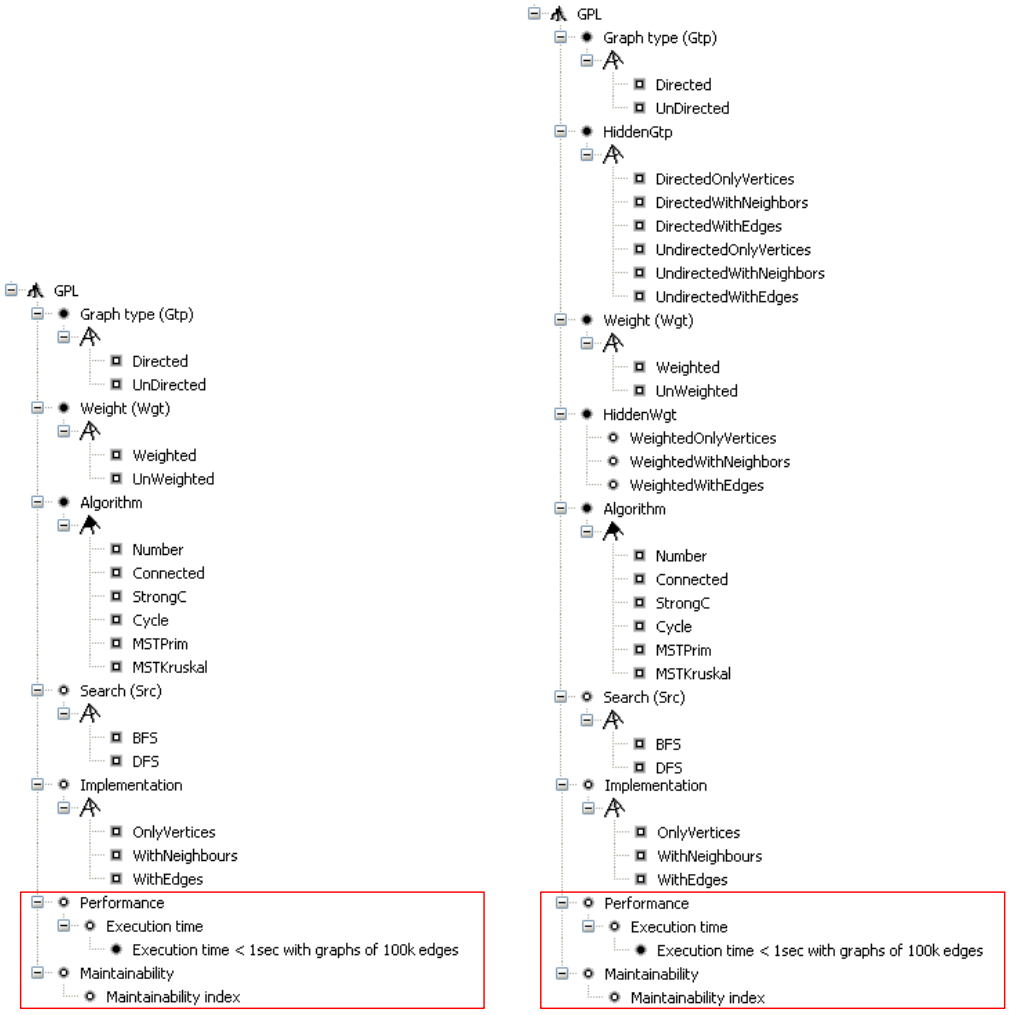


Figura I.1: Modelo de CaFM de GPL en diseño e implementación

Primeramente se han definido los impactos cualitativos. Todas las características impactan en la mantenibilidad. Y por otro lado, el diseñador con ayuda de expertos puede deducir qué características van a impactar en el tiempo de ejecución: la forma de implementar y el algoritmo principalmente. Algunas implementaciones son más adecuadas para algoritmos que trabajan con grafos con pesos. Por ejemplo, utilizar *OnlyVertices* sin representación explícita de las aristas supone más trabajo: crear las aristas utilizando la lista de adyacencia o la de vecinos que utilizar *WithEdges* donde las aristas con anotaciones de los pesos son representados explícitamente. Por otro lado, los algoritmos que no utilizan pesos tales como *Number*, *Cycle*, *Connected* o *StrongC* trabajan más rápido con menos clases para leer y trabajar con ellos.

Impactos cuantitativos:

- Number + WithEdges impact ++ Execution Time
- Number + WithNeighbors impact + Execution Time
- Cycle + WithEdges impact ++ Execution Time
- Cycle + WithNeighbors impact + Execution Time
- Connected + WithEdges impact ++ Execution Time
- Connected + WithNeighbors impact + Execution Time
- StrongC + WithEdges impact ++ Execution Time
- StrongC + WithNeighbors impact + Execution Time
- MSTPrim + OnlyVertices impact ++ Execution Time
- MSTPrim + WithNeighbors impact ++ Execution Time
- MSTKruskal + OnlyVertices impact ++ Execution Time
- MSTKruskal + WithNeighbors impact ++ Execution Time
- NoSrc impact ++ Execution Time
- All features impacts on Maintainability (La mantenibilidad se va a medir con el índice de mantenibilidad que se obtiene a partir de mediciones en el código, como todas las características añaden código, todas va a impactar en la mantenibilidad)

I.3. Validación de calidad

Una vez que los impactos cualitativos han sido determinados, la validación de la calidad puede realizarse. En esta fase además se detectarán las interacciones.

Execution time

Antes de aplicar el algoritmo para seleccionar los productos, hay que realizar algunos pasos:

Normalizar modelo de características

Transformar o normalizar el modelo de características en una lista plana de conjuntos de opciones o variantes que afectan a los aspectos de calidad seleccionados previamente.

En este caso, las características que afectan al tiempo de ejecución son tipo de algoritmo e implementación principalmente. También *Search* (Src) podría tener un impacto. El tipo de grafo (gtp y wgt) que viene en gran medida determinado por los algoritmos seleccionados también puede tener su efecto.

Primeramente hay que transformar el modelo en una lista plana. La característica tipo *Or* del algoritmo debido a todas las características hijas, tiene una combinatoria de 64 posibles alternativas. Para simplificar se ha optado por considerarla como una característica alternativa, por lo tanto sólo se consideran los productos con un solo algoritmo, y se supone que como los algoritmos se ejecutan consecutivamente no hay interacciones entre ellos.

Algorithm = {Number, Connected, StrongC, Cycle, MSTPrim, MSTKruskal}

Implementation = {OnlyVertices, WithNeighbours, WithEdges}

Src = {NoSrc, BFS, DFS}

Gtp = {Directed, Undirected}

Wgt = {Weighted, UnWeighted}

Eliminar las dependencias

Las dependencias son:

- Number excludes NoSrc
- Connected excludes NoSrc
- Connected requires Undirected
- StrongC requires DFS
- StrongC requires Directed

- Cycle requires DFS
- MSTPrim requires Undirected
- MSTPrim requires Weighted
- MSTKruskal requires Undirected
- MSTKruskal requires Weighted
- MSTKruskal requires WithEdges
- MSTKruskal excludes MSTPrim

Tras eliminar las dependencias se obtienen 6 listas.

Algorithm = {Number } Implementation = {OnlyVertices, WithNeighbours, WithEdges } Src = {BFS, DFS } Gtp = {Directed, Undirected } Wgt = {Weighted, UnWeighted }
Algorithm = {Connected } Implementation = {OnlyVertices, WithNeighbours, WithEdges } Src = {BFS, DFS } Gtp = {Undirected } Wgt = {Weighted, UnWeighted }
Algorithm = {StrongC } Implementation = {OnlyVertices, WithNeighbours, WithEdges } Src = {DFS } Gtp = {Directed } Wgt = {Weighted, UnWeighted }
Algorithm = {Cycle } Implementation = {OnlyVertices, WithNeighbours, WithEdges } Src = {DFS } Gtp = {Directed, Undirected } Wgt = {Weighted, UnWeighted }
Algorithm = {MSTPrim } Implementation = {OnlyVertices, WithNeighbours, WithEdges } Src = {NoSrc, BFS, DFS } Gtp = {Undirected } Wgt = {Weighted }
Algorithm = {MSTKruskal } Implementation = {WithEdges } Src = {NoSrc, BFS, DFS } Gtp = {Undirected } Wgt = {Weighted }

Aplicar el algoritmo de selección de productos

Se calculan los productos a evaluar en cada lista para detectar interacciones grado 2: en total son 44 productos de un total de 160 (27,5 % del total de productos) y se aplica el algoritmo para seleccionar los productos.

Para medir el tiempo de ejecución, se han ejecutado los productos con gráficos generados aleatoriamente y con 1000 vértices y 50k, 100k, 150k, 200k, 250k y 300k aristas. La medición se ha realizado 5 veces y se ha seleccionado el valor que más se ha repetido por cada producto (ver resultados en Tabla I.1).

El escenario obligatorio: "*Execution time<1sec with graphs of 100k edges*" se cumple en todos los casos para ese tamaño de grafo. En grafos mayores no siempre se cumple. De los productos evaluados, el producto con peor tiempo (el producto 39) tiene un tiempo de ejecución de 500 milisegundos. MSTPrim es el algoritmo que más tiempo requiere y *NoSrc* (no tener un algoritmo de

Tabla I.1: Productos y resultados de la medición de los productos en milisegundos

Num	0	1	2	3	4	50k	100k	150k	200k	250k	300k
0	Weighted	Directed	OnlyVertices	BFS	Number	16	94	47	109	78	16
1	Weighted	Directed	WithNeighbors	BFS	Number	15	47	109	62	188	15
2	Weighted	Directed	WithEdges	BFS	Number	31	46	47	78	281	31
3	Unweighted	Undirected	WithNeighbors	DFS	Number	31	47	63	78	110	31
4	Unweighted	Undirected	OnlyVertices	DFS	Number	16	47	47	78	78	16
5	Unweighted	Undirected	WithEdges	DFS	Number	16	47	63	94	109	16
6	Weighted	Directed	OnlyVertices	DFS	Number	16	15	31	46	47	16
7	Unweighted	Undirected	OnlyVertices	BFS	Number	31	125	94	219	125	31
8	Weighted	Undirected	OnlyVertices	BFS	Number	31	125	94	219	140	31
9	Unweighted	Directed	OnlyVertices	BFS	Number	16	63	47	110	94	16
10	Weighted	Undirected	OnlyVertices	BFS	Connected	31	125	93	218	250	31
11	Weighted	Undirected	WithNeighbors	BFS	Connected	31	63	78	313	328	31
12	Weighted	Undirected	WithEdges	BFS	Connected	110	203	312	125	141	110
13	Unweighted	Undirected	WithNeighbors	DFS	Connected	31	62	63	78	109	31
14	Unweighted	Undirected	OnlyVertices	DFS	Connected	31	32	47	62	78	31
15	Unweighted	Undirected	WithEdges	DFS	Connected	31	47	78	79	109	31
16	Weighted	Undirected	OnlyVertices	DFS	Connected	15	31	47	78	78	15
17	Unweighted	Undirected	OnlyVertices	BFS	Connected	15	125	78	219	110	15
18	Weighted	Directed	OnlyVertices	DFS	StrongC	47	125	203	219	344	47
19	Weighted	Directed	WithNeighbors	DFS	StrongC	109	187	391	344	578	109
20	Weighted	Directed	WithEdges	DFS	StrongC	172	313	500	563	1016	172
21	Unweighted	Directed	WithNeighbors	DFS	StrongC	125	203	360	328	406	125
22	Unweighted	Directed	OnlyVertices	DFS	StrongC	47	125	203	219	313	47
23	Unweighted	Directed	WithEdges	DFS	StrongC	172	297	484	563	813	172
24	Weighted	Directed	OnlyVertices	DFS	Cycle	16	16	31	31	47	16
25	Weighted	Directed	WithNeighbors	DFS	Cycle	32	15	47	47	47	32
26	Weighted	Directed	WithEdges	DFS	Cycle	16	31	32	63	63	16
27	Unweighted	Undirected	WithNeighbors	DFS	Cycle	31	47	63	78	109	31
28	Unweighted	Undirected	OnlyVertices	DFS	Cycle	31	31	63	78	94	31
29	Unweighted	Undirected	WithEdges	DFS	Cycle	16	47	63	94	109	16
30	Weighted	Undirected	OnlyVertices	DFS	Cycle	31	47	47	62	93	31
31	Unweighted	Directed	OnlyVertices	DFS	Cycle	16	15	31	47	47	16
32	Weighted	Undirected	OnlyVertices	NoSrc	MSTPrim	94	250	485	797	1360	94
33	Weighted	Undirected	OnlyVertices	BFS	MSTPrim	94	234	500	813	1344	94
34	Weighted	Undirected	OnlyVertices	DFS	MSTPrim	94	250	484	813	1359	94
35	Weighted	Undirected	WithNeighbors	BFS	MSTPrim	125	266	500	907	1594	125
36	Weighted	Undirected	WithNeighbors	NoSrc	MSTPrim	109	281	500	922	1563	109
37	Weighted	Undirected	WithNeighbors	DFS	MSTPrim	109	266	500	907	1610	109
38	Weighted	Undirected	WithEdges	DFS	MSTPrim	188	484	657	1110	1625	188
39	Weighted	Undirected	WithEdges	NoSrc	MSTPrim	172	500	656	1047	1625	172
40	Weighted	Undirected	WithEdges	BFS	MSTPrim	187	484	656	1062	1609	187
41	Weighted	Undirected	WithEdges	NoSrc	MSTKruskal	78	297	218	297	359	78
42	Weighted	Undirected	WithEdges	BFS	MSTKruskal	78	297	219	297	359	78
43	Weighted	Undirected	WithEdges	DFS	MSTKruskal	78	297	219	281	344	78

búsqueda) implica más tiempo de ejecución. Así mismo, la implemetación *WithEdges* sería el que más aumenta el tiempo de ejecución.

Como es posible que una aplicación tenga más de un algoritmo que ejecute consecutivamente, el producto límite será el que además de MSTPrim ejecuta otros algoritmos:

Weighted	Undirected	WithEdges	DFS	Number	Connected	Cycle	MSTPrim

Realmente observando los tiempos de ejecución de los productos con los algoritmos Number, Connected y Cycle se puede deducir que la suma de todos los tiempos no va a superar el segundo. De todas formas para asegurarse se pueden generar el producto en cuestión y medir el tiempo que se obtiene (620 milisegundos).

Analizando los resultados se detectan muchas interacciones, y sobre todo algo que puede sorprender: El tiempo de ejecución es siempre más alto con la implementación *WithEdges*, independientemente del algoritmo (al menos para los tamaños de grafos con los que se ha medido). Era lógico pensar que los algoritmos que trabajan con pesos, por ejemplo *MSTPrim* iban a tener mejores resultados con *WithEdges*, pero en este caso no es así. El tiempo de ejecución es superior en los productos con *WithEdges* para los tamaños de grafos mencionados. Esto sucede para todos los algoritmos y no sólo para *Number*, *Cycle*, *Connected*, *StrongC*, etc.

I.4. Derivación

Tiempo de ejecución

En este caso y teniendo en cuenta que no es posible optimizar el tiempo de ejecución de los productos durante la derivación seleccionando la implementación más adecuada para los algoritmos incluidos; no se considera necesario tener en cuenta el tiempo de ejecución durante la derivación, ya que dependerá más del algoritmo y del tipo de grafo seleccionado (que normalmente viene marcado por la funcionalidad requerida).

Mantenibilidad

En el caso de la mantenibilidad, cada característica impacta en este atributo.

Para medir el mantenimiento, se ha utilizado la métrica *maintainability index*. Esta métrica fue propuesta por el SEI [SEI 07b] y ha sido validada en diferentes trabajos [Pearse 95] [Zhuo 93]. A partir del *maintainability index* de los productos se puede inferir el *maintainability index* de cada característica (como se propone en [Aldekoa 08]). *Aldekoa et al* proponen generar y medir todos los productos para obtener el *maintainability index* de los productos. En nuestro caso hemos aplicado el método y tras medir 44 productos se han obtenido los impactos. Estos impactos no son el *maintainability index* de las características como se obtienen en [Aldekoa 08] tras medir todos los productos. Los impactos son incrementales, cuánto aumenta o disminuye el *maintainability index* al cambiar de una variante a otra. Esta información también puede ser de utilidad a la hora de realizar el mantenimiento y para conocer el *maintainability index* de los productos durante la derivación siempre y cuando este sea un dato importante a nivel de producto.

Se han detectado varias interacciones, el tipo de implementación con *Gtp* y con *Wgt*. Estas interacciones eran previsibles teniendo en cuenta la implementación de la línea que se observa en la figura I.1.

A continuación se muestran algunos de los impactos cuantitativos en *maintainability index*:

Características	Impacto
$OnlyVertices \rightarrow WithNeighbours(Weighted, Directed)$	-6.79
$OnlyVertices \rightarrow WithNeighbours(Weighted, Undirected)$	-2.72
$OnlyVertices \rightarrow WithNeighbours(Unweighted, Directed)$	-8.32
$OnlyVertices \rightarrow WithNeighbours(Unweighted, Undirected)$	-3.59
$BFS \rightarrow DFS(NumberorConnected)$	0,76
$BFS \rightarrow DFS(MSTPrim)$	0,56
$BFS \rightarrow DFS(MSTPrim)$	0,56
...	

I.5. Enfoque *bottom-up*

Se han generado todos los productos y se ha medido el tiempo de ejecución. El producto límite respecto al tiempo, es decir, el producto que más tiempo requiere es el mismo que se había detectado mediante el método.

I.6. Conclusiones

Características evaluación				Resultados			
Atributos evaluados	Fase de evaluación	Estrategia de reducción	Técnicas de evaluación	N° total productos	N° productos evaluados	Reducción esfuerzo	Grado de confianza
Rendimiento Mantenibilidad	Implementación	Selección de un subconjunto de productos	Medición de los aspectos de calidad mediante ejecución de los productos	160	44	72,5 %	Alto

El método está orientado principalmente a evaluar atributos de calidad operacionales que pueden ser variantes de un producto a otro, pero también se puede utilizar para atributos de calidad de desarrollo importantes en una línea de productos tales como la mantenibilidad. El modelo de características extendido también puede ser utilizado para mejorar la calidad a nivel de línea de productos. En este caso, los valores de *maintainability index* de los productos pueden ayudan a identificar las variantes que disminuyen la mantenibilidad. Esta información es muy útil durante el mantenimiento preventivo para centrarse en esas características y mejorar su *maintainability index* mediante refactorización. Y la información también puede ser utilizada durante derivación si la mantenibilidad es importante a nivel de producto.

El análisis de los resultados de las mediciones ayuda a corregir asunciones incorrectas realizadas por los diseñadores a la hora de especificar los impactos. En este caso, se suponía que un diseño podría ofrecer mejores resultados de tiempo con algunos algoritmos pero la experimentación ha demostrado que no es así, al menos para los tamaños de grafos con las que se ha probado.

En este caso de estudio el número de productos a evaluar es superior que en el resto de los casos de estudio porque todas las características afectan a los atributos de calidad elegidos para evaluar.