eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Aplicación de una arquitectura de Líneas de Producto para una familia de anotadores Web

Autor:

Xabier Garmendia Díaz

Dirigido por:

Óscar Díaz

Resumen

En este documento se recoge la memoria del Trabajo de Fin de Grado “Aplicación de una arquitectura de Líneas de Producto para una familia de Anotadores Web” realizado por Xabier Garmendia Díaz para la titulación de Grado en Ingeniería Informática en la especialidad de Ingeniería del Software, en la Facultad de Informática de la Universidad del País Vasco (UPV/EHU) de Donostia-San Sebastián.

El proyecto ha sido dirigido por el profesor de la UPV/EHU Óscar Díaz García y la colaboración de Haritz Medina Camacho, ambos miembros del grupo de investigación *Onekin* con sede en la Facultad de Informatica de la UPV/EHU en San Sebastián.

El objetivo principal de este proyecto es realizar a través del entorno de desarrollo *pure::variants* una *Línea de Producto Software* sobre un dominio de aplicaciones que realizan anotaciones Web a través de extensiones de *Google Chrome*.

Índice general

Resumen I

Índice general III

Índice de figuras VII

Índice de tablas XI

1. Introducción 1

1.1. Antecedentes del proyecto 3

1.2. Estructura de la memoria 4

2. Planificación inicial del proyecto 7

2.1. Objetivos del proyecto 7

2.2. Alcance del proyecto 8

2.3. La estructura de descomposición del trabajo (EDT) 10

2.4. Estimación de dedicación 13

2.5. Diagrama de Gantt 15

2.6. Herramientas 18

2.6.1. Draw.io 18

2.6.2. GanttProject 18

2.6.3. GoogleDrive 18
2.6.4. DiffMerge 18
2.6.5. Review&Go 19
2.6.6. GitHub 19
2.6.7. IntelliJ IDEA 19
2.6.8. *Pure::variants* 19
2.6.9. TextMaker 20
2.6.10. Google Docs 20
2.6.11. Presentaciones de Google 20
2.7. Interesados 20
2.8. Gestión de riesgos 21
2.9. Sistemas de información y comunicación 23

3. Dominio de anotaciones Web **25**
3.1. Anotaciones Web 25
3.2. Estándar W3C de anotaciones web 26
3.3. Hypothes.is 29

4. Líneas de Productos Software y *pure::variants* **33**
4.1. Líneas de Productos Software 33
4.1.1. Beneficios 36
4.1.2. Estrategias para iniciar una LPS 38
4.1.3. Ingeniería de Dominio 39
4.1.4. Ingeniería de Producto 39
4.2. Pure::variants 40
4.2.1. Feature Model 40
4.2.2. Family Model 44
4.2.3. Etiquetación de ficheros 46
4.2.4. Generación de productos. VDM 50

5. Productos de partida **53**

5.1. Highlight&Go . . . 53

5.2. Mark&Go . . . 57

5.3. Review&Go . . . 60

5.4. Arquitectura de las aplicaciones . . . 63

6. Análisis y diseño de la LPS: **73**

6.1. Analisis del dominio . . . 77

6.2. Storage . . . 78

6.3. Definition . . . 80

6.4. Production . . . 83

6.5. Consumption . . . 85

6.6. GroupManipulation . . . 88

7. Implementación **91**

7.1. Preparación del entorno de trabajo . . . 91

7.2. Implementación de *Features* . . . 98

7.3. Refactorización de *Features* . . . 101

7.3.1. Nuevo modelo de datos . . . 101

7.3.2. Añadir botones dinámicamente (Dynamic) . . . 102

7.3.3. Generación de hoja de cálculo (GSheetConsumer) . . . 103

7.3.4. Validar una anotación (Validate) . . . 103

7.4. Obtención de productos a partir de *Features* . . . 104

7.5. Resultados del proyecto . . . 106

7.5.1. Storage . . . 107

7.5.2. Definition . . . 108

7.5.3. Production . . . 110

7.5.4. Consumption . . . 111

7.5.5. GroupManipulation . . . 112

8. Seguimiento del proyecto **117**

8.1. Dedicación al proyecto 117

8.2. Soluciones a riesgos y gestión de calidad 120

8.3. Diagrama Gantt final 122

9. Conclusiones **125**

9.1. Conclusión del proyecto 125

9.2. Conocimientos adquiridos y experiencia personal 127

9.3. Mejoras y líneas futuras 128

Anexos

A. Actas de reuniones **131**

A.1. Acta 1 132

A.2. Acta 2 134

A.3. Acta 3 136

A.4. Acta 4 138

A.5. Acta 5 139

A.6. Acta 6 140

A.7. Acta 7 141

A.8. Acta 8 142

A.9. Acta 9 143

A.10. Acta 10 144

A.11. Acta 11 145

B. Manual de instalación de los productos generados **147**

Bibliografía **153**

Índice de figuras

1.1. Representación de cómo se compondrá la LPS. 4

2.1. Representación del solapamiento de las funcionalidades. 8
2.2. EDT del proyecto. 11
2.3. Parte del diagrama Gantt (22 de enero - 22 de febrero). 15
2.4. Parte del diagrama Gantt (23 de febrero - 26 de marzo). 16
2.5. Parte del diagrama Gantt (27 de marzo - 29 de abril). 16
2.6. Parte del diagrama Gantt (30 de abril - 31 de mayo). 17
2.7. Parte del diagrama Gantt (1 de junio - 1 de julio). 17

3.1. Ilustración de la idea de la capa de anotación 26
3.2. Modelo de datos de una anotación web. 27
3.3. Ejemplo de una anotación web siguiendo el estándar W3C. 29
3.4. Realización de anotaciones con la extensión de *Hypothes.is* 30
3.5. Interfaz de *Hypothes.is* donde ver los grupos y anotaciones. 31

4.1. Producción siguiendo el método *clone&own*. 34
4.2. Producción siguiendo la arquitectura de Línea de Producto. 36
4.3. Comparación de costes: Producción convencional vs LPS. 37
4.4. *Mandatory Feature* y *Optional Feature*. 41

4.5. *Alternative Features*. 41

4.6. *Or Features*. 42

4.7. Ejemplo de un *Feature Model*. 42

4.8. Ejemplo de un *Feature Model* en *pure::variants*. 43

4.9. Ejemplo de un *Feature Model* en *pure::variants* incluyendo una restricción. 43

4.10. Ejemplo de un *Family Model* en *pure::variants*. 45

4.11. Ejemplo de la selección de *Features* en un fichero VDM. 51

5.1. *Classifying mode* de *Highlight&Go*. 54

5.2. *Checking mode* de *Highlight&Go*. 55

5.3. Vista generada en la hoja de cálculo a través de las anotaciones en *Highlight&Go*. 56

5.4. Generación del *highlighter* a partir de la rúbrica. 58

5.5. Resultados plasmados de la corrección en Moodle. 59

5.6. Interfaz de *Review&Go* a la hora de realizar la revisión. 60

5.7. Comentar sobre una anotación en *Review&Go*. 61

5.8. Ilustración de las funcionalidades encontradas en los productos de partida. 62

5.9. Arquitectura de una extensión web. 64

5.10. Contenido de la carpeta de la aplicación 65

6.1. Comparación entre carpetas en DiffMerge. 74

6.2. Comparación entre ficheros en DiffMerge. 74

6.3. Extracción de *Features* de un fichero utilizando *Review&Go*. 75

6.4. Ejemplo de la organización de una de las carpetas de uno de los productos. 76

6.5. Ejemplo del contenido de las dos versiones PDF de uno de los ficheros de los productos de partida 76

6.6. Primer nivel del Modelo de Características de anotadores Web. 77

6.7. *Features* relacionadas con el Storage del Modelo de Características de anotadores Web. 79

6.8. *Highlighter* que permite etiquetar en base a Themes. 81

6.9. *highlighter* que permite etiquetar en base a Themes y Codes. 81

6.10. *Features* relacionadas con Definition del Modelo de Características de anotadores Web. 82

6.11. Ejemplo del menú de selección de la operación que se quiere realizar sobre la anotación. 83

6.12. Advertencia que se lanza con SentimentAnalysis si el comentario es ofensivo. 84

6.13. Interfaz de sugerencia de comentarios previamente utilizados, *Feature* Autofill. 84

6.14. *Features* relacionadas con Production del Modelo de Características de anotadores Web. 85

6.15. De izquierda a derecha, los botones que realizan la función de Screenshot, Canvas y TextSummary. 87

6.16. Ejemplo de lo que se visualiza mediante el tooltip al posicionarse sobre una anotación Web. 87

6.17. Visualización de las anotaciones a través de una vista Canvas. 87

6.18. Sistema de filtrado de anotaciones en base a usuarios. 88

6.19. *Features* relacionadas con Consumption del Modelo de Características de anotadores Web. 88

6.20. Lista desplegable de la *Feature* Manual. 89

6.21. Botón con el que se realiza la acción de borrar todas las anotaciones. . . . 89

6.22. *Features* relacionadas con GroupAnnotation del Modelo de Características de anotadores Web. 90

7.1. Actividades a realizar para preparar el entorno de trabajo. 92

7.2. Ficheros que se deben configurar para realizar la LPS. 92

7.3. Parte del *Feature Model* del proyecto en *pure::variants* 93
7.4. Estructura y *Family Model* del proyecto. 94
7.5. Carpeta del proyecto con el código fuente y su mapeo en el *Family Model*. 95
7.6. Configuración de la lista de modelos del *Configuration Space* del proyecto. 96
7.7. Configuración de los Input-Output del *Configuration Space* del proyecto. 97
7.8. Configuración de las transformaciones VDM del *Configuration Space* del proyecto. 97
7.9. Actividades a realizar para preparar el entorno de trabajo. 98
7.10. Parte del fichero Toolset.js donde se muestra código etiquetado. 99
7.11. Nuevo modelo de datos para realizar anotaciones. 102
7.12. Implementación para la *Feature* Reply. 104
7.13. Implementación para la Feature Validate. 104
7.14. *Features* seleccionadas para reconstruir *Highlight&Go*. 105
7.15. *Features* seleccionadas para reconstruir *Review&Go*. 106
7.16. Producto generado a través de la LPS. 114

8.1. Parte del diagrama Gantt (1 de marzo - 1 de abril). 123
8.2. Parte del diagrama Gantt (1 de abril - 1 de mayo). 123
8.3. Parte del diagrama Gantt (1 de mayo - 1 de junio). 124
8.4. Parte del diagrama Gantt (1 de junio - 1 de julio). 124

B.1. Carpeta generada por la LPS que pertenece a un producto. 148
B.2. Ejecución del comando *npm install*. 148
B.3. El estado de la carpeta tras ejecutar el comando *npm install*. 149
B.4. El estado de la carpeta tras ejecutar el comando *gulp default*. 149
B.5. Carpetas creadas tras la ejecución de *gulp default* en la que se encuentran los ficheros de ejecución. 150
B.6. Pestaña que se debe abrir para que se muestren las extensiones. 150
B.7. Botón que se debe pulsar para instalar la carpeta de la extensión. 151

Índice de tablas

2.1. Descripciones de las tareas del proyecto. 13

2.2. Dedicación estimada para cada tarea. 14

4.1. Sentencias de *pure::variants* para asociar bloques de código con las *Features*. 48

5.1. Tipos de archivos que guarda cada carpeta de la aplicación. 65

5.2. Ficheros de la carpeta images. 66

5.3. Ficheros de la carpeta pages. 67

5.4. Ficheros .js de la carpeta scripts. 68

5.5. Ficheros de la carpeta scripts/background. 68

5.6. Fichero de la carpeta scripts/hypothesis. 68

5.7. Ficheros de la carpeta scripts/model. 69

5.8. Ficheros de la carpeta scripts/contentScript. 69

5.9. Ficheros de la carpeta styles. 71

6.1. Las *Features* que se agrupan dentro de Target. 80

6.2. Las *Features* que se agrupan dentro de Production. 83

6.3. Las *Features* que se agrupan dentro de Comment. 84

6.4. Las *Features* que se agrupan dentro de Production. 86

7.1. Composición de las *Features* que se agrupan en Storage. 107

7.2. Composición de las *Features* que se agrupan en Target 108

7.3. Composición de las *Features* que se agrupan en Tag-based body. 109

7.4. Composición de las *Features* que se agrupan en Production. 110

7.5. Composición de las *Features* que se agrupan en Consumption. 111

7.6. Composición de las *Features* que se agrupan en GroupManipulation. . . . 112

7.7. Restricciones establecidas en el *Feature Model*. 114

8.1. Comparativa entre la dedicación estimada y real. 119

1. CAPÍTULO

Introducción

Hoy en día es muy común encontrar empresas que ofrecen su producto software a múltiples clientes y, como consecuencia de ello, es posible que cada cliente busque variaciones distintas del producto que lo personalicen para su propio uso acorde a sus necesidades. A raíz de las constantes peticiones de variación mencionadas anteriormente, nacen las familias de productos software. Una familia de productos está compuesta por productos que reúnen elementos comunes, pero se diferencian por las distintas variaciones que se les implementa para personalizarlos.

El concepto de una familia de productos no está necesariamente relacionado con el software. Este hecho también se puede dar en distintos ámbitos, como por ejemplo en la producción de automóviles. Una empresa de automóviles al crear un nuevo producto (en este caso, un coche) a su vez crea distintos modelos, y estos nuevos coches forman lo que sería la familia de productos. Los coches creados por la empresa tendrán en común varios elementos, pero luego pueden surgir diferentes variaciones a partir del color del coche, el tipo de motor, el tipo de combustible que consume, etcétera; que formarían los puntos de variación.

En el caso del software, a la hora de mantener los códigos de los distintos productos que mantiene una empresa, surge el problema de gestionar las diferentes versiones y el crecimiento que puedan llegar a tener los elementos comunes que comparte la familia de productos.

La manera tradicional de realizar una distinta variación del producto principal es obtener una copia del código y realizar en esa misma copia las distintas variaciones que se desean

para personalizar el producto nuevo, sin embargo, esta técnica de personalización en serie (*mass customization*) no resulta eficiente por temas de mantenimiento del código.

En un principio, para la generación de nuevos productos utilizar la técnica mencionada anteriormente podría resultar fácil y rápida, pero a medida que evoluciona el producto y se van creando nuevas variaciones, mantener esos códigos en diferentes repositorios resulta una tarea casi imposible. Tener que realizar un cambio en uno de los elementos comunes de los productos, provoca que se deba realizar ese cambio en todos los demás.

Por lo tanto, para lidiar con el reto de crear eficientemente diferentes variaciones de un producto software, surge la idea de las *Líneas de Productos Software* (LPS).

El objetivo principal de una LPS es reunir los elementos comunes de los productos que formarán el dominio de la producción y gestionar las distintas variaciones que pueden llegar a tener los productos, pudiendo así ser reutilizables en más de uno de los productos que se generen a través de la cadena de montaje.

El objetivo de este Trabajo de Fin de Grado ha sido indagar en este enfoque de creación de nuevos productos y realizar la implementación de una LPS que de soporte a una familia de productos software. La intención es que la LPS permita gestionar las diferentes variaciones que existen entre los productos y se consiga aplicar uno de los puntos importantes de la Ingeniería del Software, la reutilización.

En el siguiente apartado 1.1 se especifica detalladamente sobre qué trata la Línea de Producto, la razón por la que se crea y sobre qué dominio se realiza. En el último apartado del presente capítulo se encuentra un resumen de la estructura de la memoria realizada.

Junto a la memoria del trabajo está disponible el código fuente de la línea de producto que se ha desarrollado a lo largo del proyecto. El código fuente del proyecto debe importarse a *pure::variants*, que es la herramienta utilizada para la definición de la LPS. Una vez importado el proyecto, ya se pueden generar productos.

El código fuente está disponible en la siguiente dirección de *GitHub*:

`https://github.com/xabigar/WebAnnotatorSPL`

1.1. Antecedentes del proyecto

El proyecto de este Trabajo de Fin de Grado parte de una familia de productos que han sido desarrollados utilizando el enfoque de *clone&own*[1]. Esto ha dado lugar a una redundancia de código donde el mismo código se mantiene en distintos repositorios y complica su mantenimiento, ya que un cambio en uno de los productos puede conllevar tener que repetirlo en cada uno de los repositorios donde se encuentre el código que se ha modificado.

La familia de productos consta de tres herramientas de anotación Web [2], cada una enfocada en un dominio distinto donde la tarea de anotar se hace con un propósito u objetivo diferente. Las herramientas son extensiones del navegador *Google Chrome*, es decir, los productos que se toman como base para realizar la LPS son programas que añaden nuevas funciones al navegador, en este caso, para realizar y gestionar anotaciones Web.

Los nombres de las extensiones de *Google Chrome* que existen para realizar el proyecto son *Highlight&Go* 5.1, *Mark&Go* 5.2 y *Review&Go* 5.3. Los tres productos han sido creados y desarrollados por Haritz Medina Camacho, que actualmente está trabajando en una tesis doctoral que se centra en la aplicación de las anotaciones Web en diferentes dominios. Los productos fueron creados en el orden que se ha mencionado anteriormente, por lo tanto, la herramienta *Mark&Go* se desarrolló a partir de una copia de *Highlight&Go* y realizando una copia de *Mark&Go* finalmente se diseñó y se creó *Review&Go*. A causa de realizar este sistema para crear los nuevos productos, en los proyectos de las herramientas se encuentran ficheros innecesarios y partes de código que no se utilizan que se han heredado del producto anterior.

Para la futura creación de nuevos productos y el mantenimiento de los actuales, se valoró aplicar una solución de Líneas de Productos Software utilizando la herramienta *pure::variants* sobre el dominio de anotaciones Web. Aparte de ayudar a mantener los actuales productos también permite crear unos nuevos mezclando las distintas variaciones que se han extraído de los actuales. Además, se evita la existencia de ficheros que no se utilizan, para ello, la LPS se encarga de introducir cada fichero en las aplicaciones cuando sea necesario.

El resultado del proyecto es una LPS que sirve de punto de partida para dar soporte a los

[1] Se trata de reutilizar partes de código ya implementadas y modificarlo para desarrollar una aplicación parecida.

[2] Una anotación web es una anotación en línea asociada con un recurso web.

tres productos actuales y además, ayudará a seguir desarrollando nuevas funcionalidades y nuevos cambios para la familia de productos 1.1. La LPS permitirá un desarrollo más ágil, mejorando el mantenimiento de los diferentes productos y permitiendo la fácil extensión de nuevas características que se puedan requerir por parte de la comunidad de anotaciones Web.

Gracias a mantener el código en un único espacio tan solo será necesario realizar las nuevas implementaciones sobre la línea de producto, en vez de tener que aplicar los cambios en cada uno de los productos en los que se desea integrar.

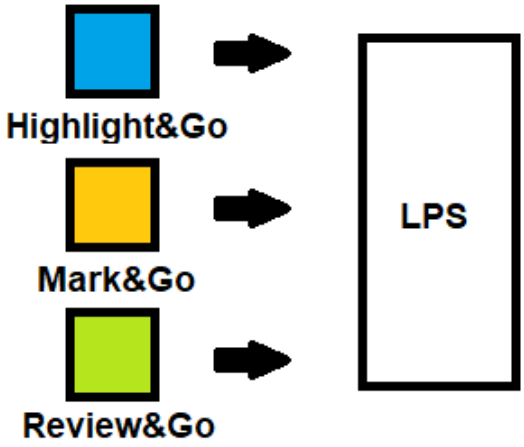


Figura 1.1: Representación de cómo se compondrá la LPS.

1.2. Estructura de la memoria

La estructura de la presente memoria es la siguiente:

1. **Introducción:** El objetivo de este capítulo es realizar una introducción al lector sobre el contenido del trabajo y describe la situación de partida en la que se enmarca el proyecto.

2. **Planificación inicial del proyecto:** Se definen los objetivos y se encuentra la planificación establecida para llevar a cabo el proyecto.

3. **Dominio de anotaciones Web:** En este apartado se explican varios conceptos relacionados con el dominio de la LPS que se ha desarrollado en este proyecto.

4. **Líneas de Productos Software y *pure::variants*:** Se detalla la arquitectura de las Líneas de Productos Software y se explica el funcionamiento de la herramienta de desarrollo *pure::variants*.

5. **Productos de partida:** En este capítulo se presentan los productos de partida que se utilizan para la realización de la LPS y su arquitectura.

6. **Análisis y diseño de la LPS:** En este capítulo se especifica el diseño para la Línea de Productos que consiste en la realización del Modelo de Características.

7. **Implementación:** Se explica cómo se ha implementado el proyecto de la LPS en *pure::variants*.

8. **Seguimiento del proyecto:** En este capítulo se muestra el seguimiento y control llevado a cabo en el proyecto.

9. **Conclusiones:** Se recogen las conclusiones finales una vez finalizado el proyecto.

Anexo A. **Actas reuniones:** Se encuentran las actas que se han realizado tras las reuniones del proyecto.

Anexo B. **Manual de instalación de los productos generados:** Un manual en el que se explica como instalar la extensión del navegador generada por la LPS en *Google Chrome*.

2. CAPÍTULO

Planificación inicial del proyecto

Este capítulo recoge la planificación planteada inicialmente con la idea de ayudar a realizar el Trabajo de Fin de Grado de una manera correcta y organizada.

En primer lugar, se definen los objetivos y el alcance del proyecto a realizar. En segundo lugar, se encuentran las secciones relacionadas con la planificación mediante la estructura de descomposición del trabajo (EDT), estimación de la dedicación necesaria y el diagrama de Gantt. En los demás apartados del capítulo se presentan las herramientas necesarias para la realización del proyecto, los interesados del proyecto, la gestión de los posibles riesgos que pueden surgir, y por último, la gestión de los sistemas de información y los sistemas de comunicación.

2.1. Objetivos del proyecto

El proyecto a realizar tiene como fin principal aplicarle una solución de Líneas de Producto a una familia de productos de anotación Web utilizando la herramienta *pure::variants*. Para llevar a cabo el objetivo que se desea alcanzar también se deben lograr los siguientes objetivos:

- Realizar un estudio sobre la arquitectura de Líneas de Producto para conocer sus líneas generales y documentarlas.
- Analizar los tres productos de partida de anotaciones Web. Entender su funcionamiento y código para identificar sus funcionalidades y poder extraer los elementos

comunes y variables que tienen entre los tres 2.1. El objetivo del análisis es realizar la definición de las Características [1] (*Features*).

- Definición del Modelo de Características (*Feature Model*) sobre el dominio de anotaciones Web para la implementación de la LPS partiendo de las *Features* definidas.
- Familiarización con la herramienta *pure::variants* para el proceso de desarrollo de la LPS.
- Realizar la refactorización de código necesario para integrar los tres productos en la LPS y conseguir encajar las funcionalidades entre ellas.
- Generación de nuevos productos a partir de la LPS desarrollada.
- Documentar el proceso de desarrollo de la LPS y el aprendizaje adquirido.

Figura 2.1: Representación del solapamiento de las funcionalidades.

2.2. Alcance del proyecto

El alcance del proyecto es conseguir una Línea de Productos Software partiendo de los tres productos de anotación Web que integre *Features* que se puedan recoger de ellos. La parte primordial del proyecto es realizar un análisis correcto de los productos y realizar un diseño del dominio de anotaciones Web logrando un Modelo de Características para luego desarrollar la LPS.

Para la realización de la LPS se tendrá en cuenta el código de las herramientas de partida y el de algunas posibles actualizaciones de los productos, pero es posible que haya

[1] Una característica es una particularidad del producto que se considera importante para describir y distinguir entre las demás características de la LPS. También se conoce como *Feature*.

que llevar a cabo una refactorización del código actual para poder integrar algunas de las funcionalidades. En el caso de tener que implementar nuevas funcionalidades, realizar grandes cambios sobre el código o necesitar conocimiento sobre desarrollo web, se contará con la colaboración de Haritz Medina, desarrollador de las aplicaciones de anotaciones Web.

La LPS será un punto de partida para seguir siendo desarrollada durante la realización de la tesis de Haritz, por lo tanto, el objetivo no es implementar todas las *Features*, pero se intentará implementar la mayor cantidad de *Features* posibles para como mínimo, poder reconstruir dos de los productos a partir de las *Features* recogidas en el *Feature Model*.

No se descarta la posibilidad de que haya que realizar cambios en el código, ya que en paralelo a este proyecto puede que los tres productos de partida sufran cambios o se les añada nuevas funcionalidades.

En cuanto a la herramienta para desarrollar la Línea de Productos, se decidió utilizar *pure::variants* por ser una de las pocas herramientas que tiene un soporte comercial, amplia documentación y el grupo de investigación donde se realiza el TFG tiene amplio conocimiento de la misma. Para realizar el Modelo de Características de la LPS se tendrán en cuenta los habituales tipos de *Features*: Características obligatorias, opcionales, or-exclusivos e inclusivos. También se tratará de establecer las restricciones que sean necesarias entre las *Features* y los elementos que componen el *Family Model*[2].

[2]La información sobre de qué se trata un *Family Model* se encuentra en el capítulo 4.2.2.

2.3. La estructura de descomposición del trabajo (EDT)

La estructura de descomposición del trabajo (EDT) es una herramienta que consiste en la descomposición jerárquica del trabajo a ser ejecutado para cumplir con los objetivos del mismo y crear los entregables requeridos. El EDT del proyecto 2.2 a desarrollar cuenta con cinco secciones para repartir los paquetes de trabajo: formación, desarrollo, pruebas, gestión y documentación.

1. **Formación:** Esta sección contiene los paquetes de trabajo relacionados con el aprendizaje requerido para poder llevar a cabo el proyecto. Entre las competencias necesarias para elaborar el proyecto se encuentran las de aprender a usar herramientas nuevas: *pure::variants* y las herramientas de anotaciones Web. También se encuentran paquetes relacionados con la ampliación de conocimientos teóricos sobre Líneas de Productos y Java Script.

2. **Desarrollo:** Esta sección agrupa los paquetes de trabajo correspondientes al desarrollo del proyecto. Se encuentran dos tareas que se repiten dos veces porque está pensado realizar el desarrollo de una manera iterativa. En primer lugar, se realiza el proceso de etiquetar dos de los productos, que consiste en realizar el análisis del código y la extracción de *Features* para después implementarlas en *pure::variants*. Finalmente, se repite el mismo proceso, pero esta vez con el tercer producto.

3. **Pruebas:** Esta sección incluye los paquetes de trabajo relacionados con las pruebas para comprobar el correcto funcionamiento de las *Features* implementadas y las correcciones finales si son necesarias para finalmente realizar la validación final.

4. **Gestión:** Esta sección contiene los paquetes de trabajo relacionados con la gestión del proyecto. Entre los paquetes se encuentran la definición de objetivos, las reuniones tanto con el tutor como con el desarrollador de las herramientas, el seguimiento y la planificación.

5. **Documentación:** Esta sección incluye los paquetes de trabajo relacionados con la documentación del trabajo: la redacción de la memoria del proyecto y la preparación de la defensa.

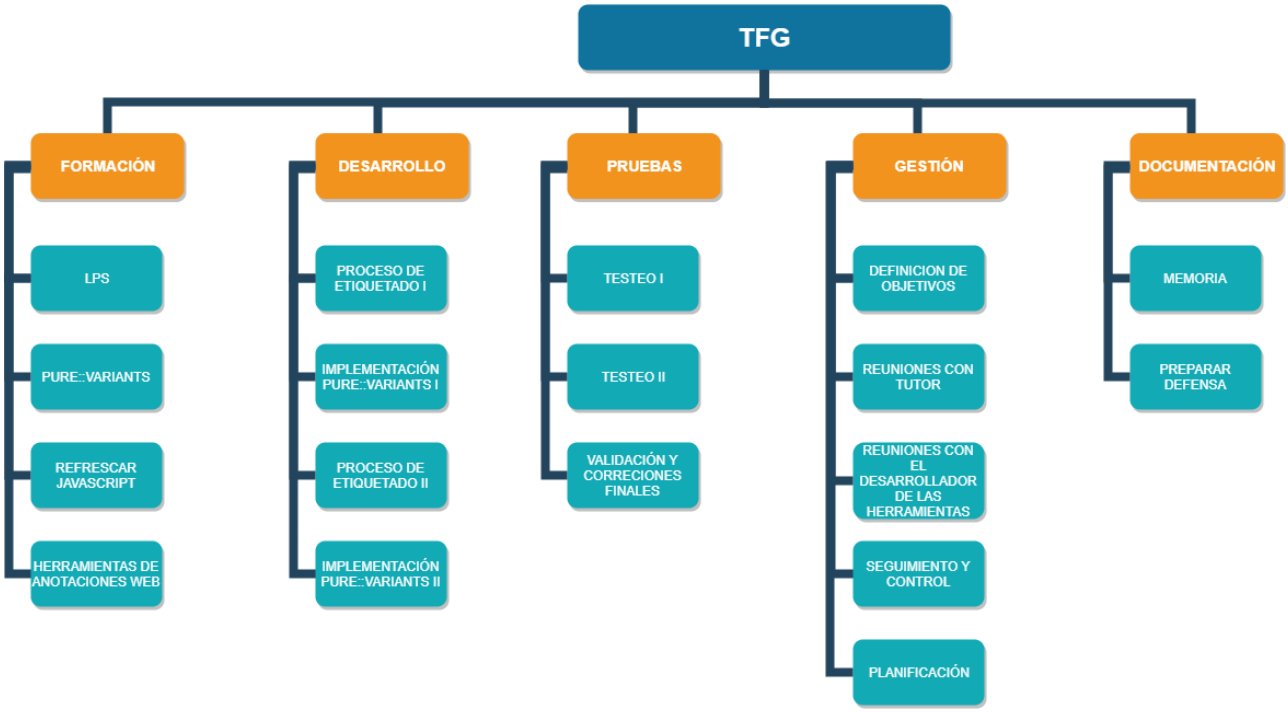


Figura 2.2: EDT del proyecto.

A continuación, en la tabla 2.1 se encuentran todos los paquetes de trabajos con sus respectivas descripciones.

Tarea	**Descripción**
	Formación
Formación sobre LPS	Ampliación de conocimientos teóricos sobre el enfoque de Líneas de Productos Software.
Pure::variants	Conocer y probar la herramienta *pure::variants* con el objetivo de familiarizarse con ella: documentación, realizar ejemplos, consultar a compañeros...
JavaScript	Repasar y aprender nuevas funcionalidades del lenguaje JavaScript para comprender el código de los productos de anotación Web. Por ejemplo, comprender cómo funcionan los *callback* [3], cómo se implementan y con qué finalidad.
Herramientas de anotaciones web	Familiarizarse con las herramientas de anotaciones Web sobre las que se realizará la LPS para aprender sus funcionalidades.
	Desarrollo
Proceso de etiquetado I	Entender, analizar y etiquetar el código de dos de las herramientas de la línea de productos: *Hightlight&Go* y *Review&Go*. El resultado debe ser un Modelo de Características
Implementación *pure::variants* I	Configurar e implementar en *pure::variants* la Línea de Productos sobre *Hightlight&Go* y *Review&Go*.
Proceso de etiquetado II	Entender, analizar y etiquetar el código de las tres herramientas de la línea de productos: *Hightlight&Go*, *Review&Go* y *Mark&Go*. El resultado debe ser el Modelo de Características definitivo.
Implementación *pure::variants* II	Configurar e implementar en *pure::variants* la Línea de Productos definitiva.
	Pruebas
Testeo I	Comprobar el correcto funcionamiento de la tarea Implementación *pure::variants* I.
Testeo II	Comprobar el correcto funcionamiento de la tarea Implementación *pure::variants* II.

Sigue en la página siguiente.

[3] Son funciones que se utilizan para continuar con la ejecución del código después de que se haya completado una operación asíncrona

Tarea	Descripción
Validación y correcciones finales	Comprobar que la implementación se ha realizado correctamente mediante la creación de diferentes productos y si persisten errores, darles solución.
Gestión	
Definición de objetivos	Definir cuál será el alcance y cuáles los objetivos del proyecto a realizar.
Reuniones con el tutor	Reuniones en el despacho del tutor para hacer seguimiento del proyecto y definir las tareas a realizar.
Reuniones con el desarrollador de las herramientas	Reuniones con el desarrollador de las herramientas para resolver problemas o dudas sobre las herramientas de anotación Web.
Seguimiento y control	Establecer las acciones que se llevarán a cabo para que la realización del proyecto se realice correctamente.
Planificación	Realizar la planificación inicial del proyecto que se recogerá en la memoria.
Documentación	
Redactar memoria	Documento que recogerá toda la labor realizada durante el proyecto.
Preparar defensa	Preparar la presentación que se hará ante el tribunal sobre el trabajo realizado.

Tabla 2.1: Descripciones de las tareas del proyecto.

2.4. Estimación de dedicación

En esta sección se incluye la tabla donde se encuentran las estimaciones del tiempo que va costar realizar cada tarea. Se trata de una estimación, por lo tanto, puede que la tarea finalmente pueda llegar a costar más o menos esfuerzo. Al finalizar el proyecto, se realizará una comparación del tiempo estimado con el que realmente ha sido requerido para terminar el proyecto. Existe también la posibilidad de que durante la realización del proyecto aparezcan nuevas tareas o que alguna de ellas no se lleven a cabo. Todos los cambios producidos en la planificación se encuentran en el capítulo 8.

Tarea	**Tiempo estimado**	**Total**
Formación		
Formación sobre LPS	3 h	62 h
Pure::variants	30 h	
Formación sobre JavaScript	5 h	
Herramientas de anotaciones web	24 h	
Desarrollo		
Proceso de etiquetado I	35 h	110 h
Implementación *pure::variants* I	25 h	
Proceso de etiquetado II	30 h	
Implementación *pure::variants* II	20 h	
Pruebas		
Testeo I	17 h	45 h
Testeo II	18 h	
Validación y correcciones finales	10 h	
Gestión		
Definición de objetivos	1 h	54 h
Reuniones con el tutor	10 h	
Reuniones con el desarrollador de las herramientas	10 h	
Seguimiento y control	15 h	
Planificación	18 h	
Documentación		
Redactar memoria	80 h	100 h
Preparar defensa	20 h	
TOTAL		371 h

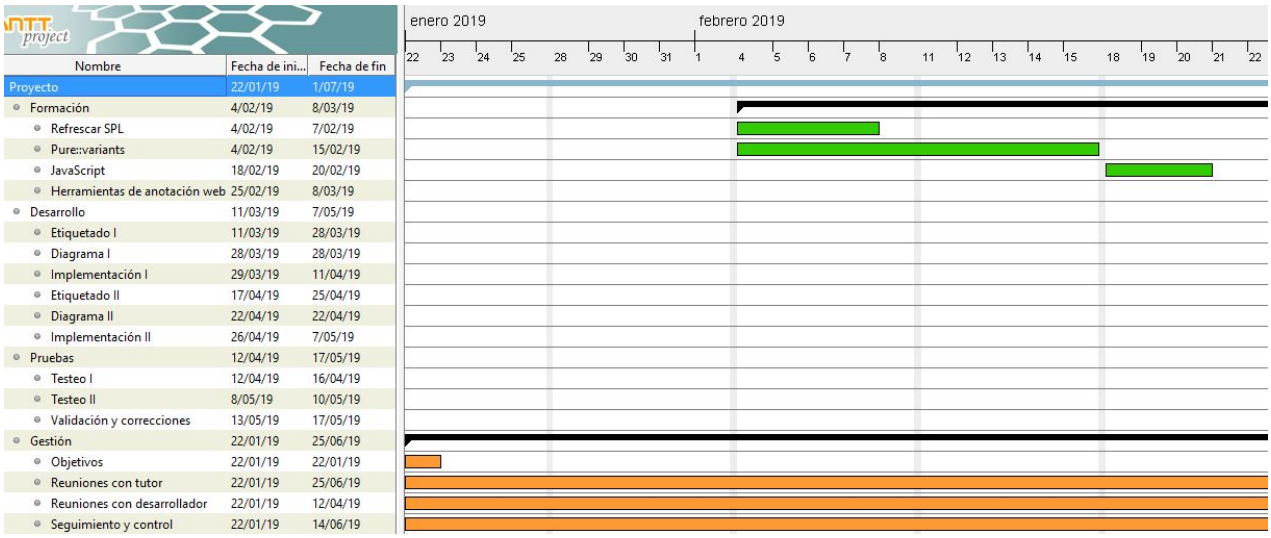
Tabla 2.2: Dedicación estimada para cada tarea.

2.5. Diagrama de Gantt

En esta sección se presenta el diagrama de Gantt inicial que se seguirá para acometer el proyecto. Se exponen los tiempos de dedicación previstos para las tareas del proyecto. En el diagrama se muestran la fecha inicial y final estimada de cada tarea. De esta manera, se puede observar cuando se pretende iniciar la tarea y que dependencias tienen entre ellas.

El proyecto empezó el 22 de enero con una reunión en la que se presentó el proyecto y las herramientas que se han de utilizar para llevarlo a cabo. Se estima que el proyecto finalice el 1 de julio, fecha en la que se espera terminar de preparar la presentación del trabajo que hay que realizar ante el tribunal.

Dentro del diagrama se encuentra la planificación de todo el proyecto, y al tratarse de un trabajo de varias semanas, el resultado es un diagrama muy extenso. Para poder apreciar bien los detalles del diagrama, se divide en las cinco figuras siguientes :



Nombre	Fecha de ini...	Fecha de fin
Proyecto	22/01/19	1/07/19
Formación	4/02/19	8/03/19
Refrescar SPL	4/02/19	7/02/19
Pure::variants	4/02/19	15/02/19
JavaScript	18/02/19	20/02/19
Herramientas de anotación web	25/02/19	8/03/19
Desarrollo	11/03/19	7/05/19
Etiquetado I	11/03/19	28/03/19
Diagrama I	28/03/19	28/03/19
Implementación I	29/03/19	11/04/19
Etiquetado II	17/04/19	25/04/19
Diagrama II	22/04/19	22/04/19
Implementación II	26/04/19	7/05/19
Pruebas	12/04/19	17/05/19
Testeo I	12/04/19	16/04/19
Testeo II	8/05/19	10/05/19
Validación y correcciones	13/05/19	17/05/19
Gestión	22/01/19	25/06/19
Objetivos	22/01/19	22/01/19
Reuniones con tutor	22/01/19	25/06/19
Reuniones con desarrollador	22/01/19	12/04/19
Seguimiento y control	22/01/19	14/06/19

Figura 2.3: Parte del diagrama Gantt (22 de enero - 22 de febrero).

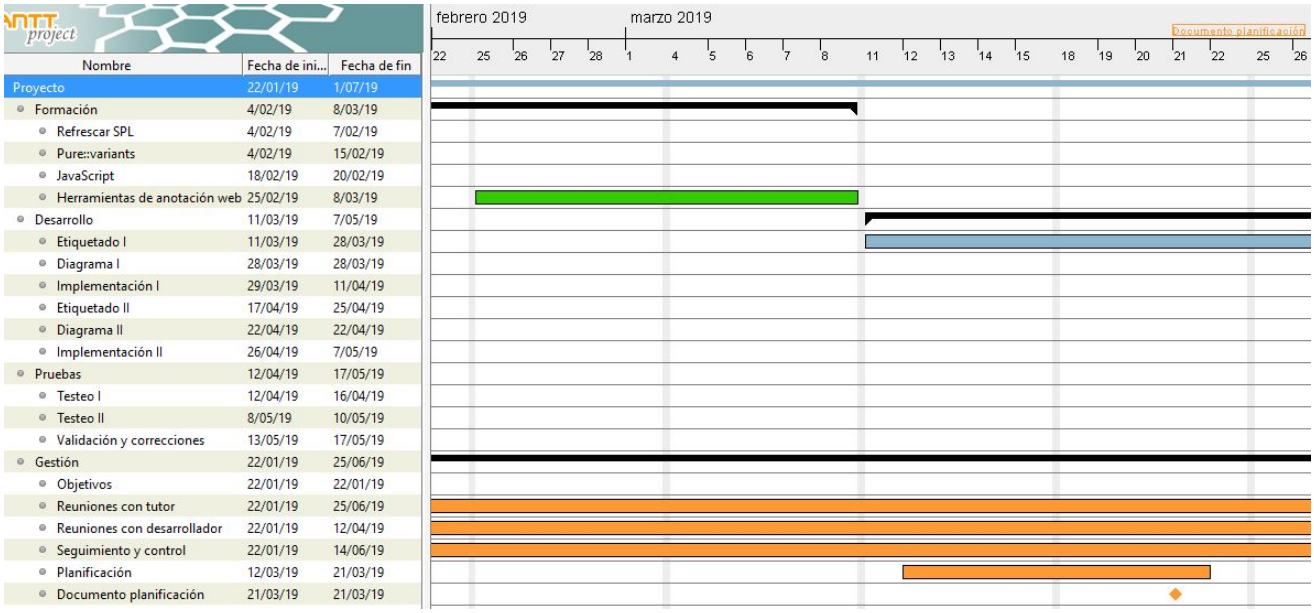


Figura 2.4: Parte del diagrama Gantt (23 de febrero - 26 de marzo).

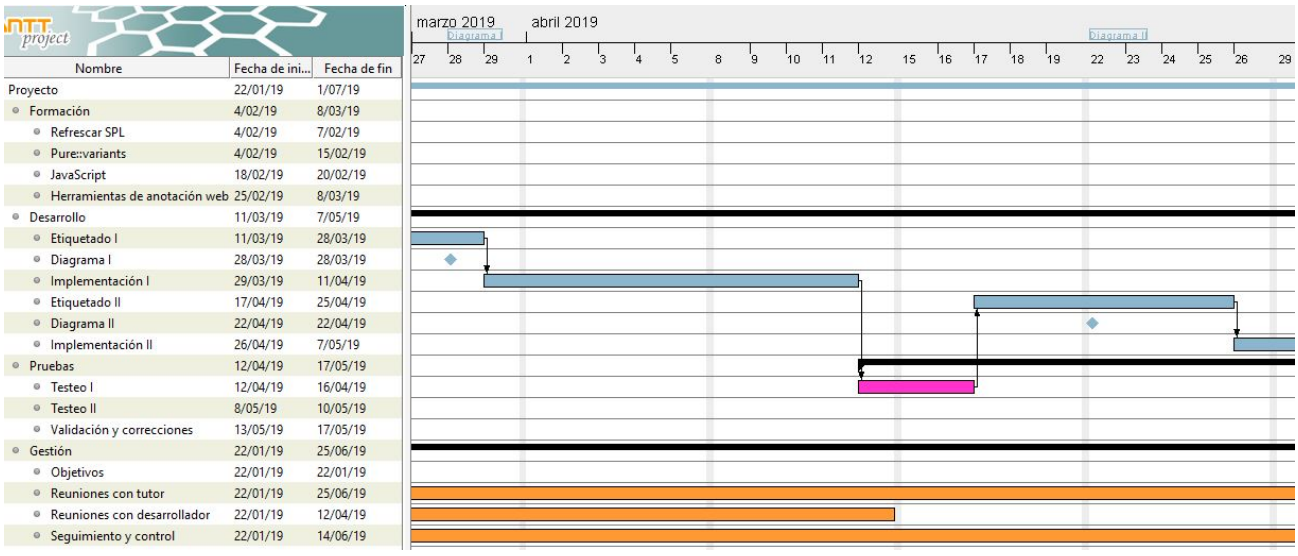


Figura 2.5: Parte del diagrama Gantt (27 de marzo - 29 de abril).

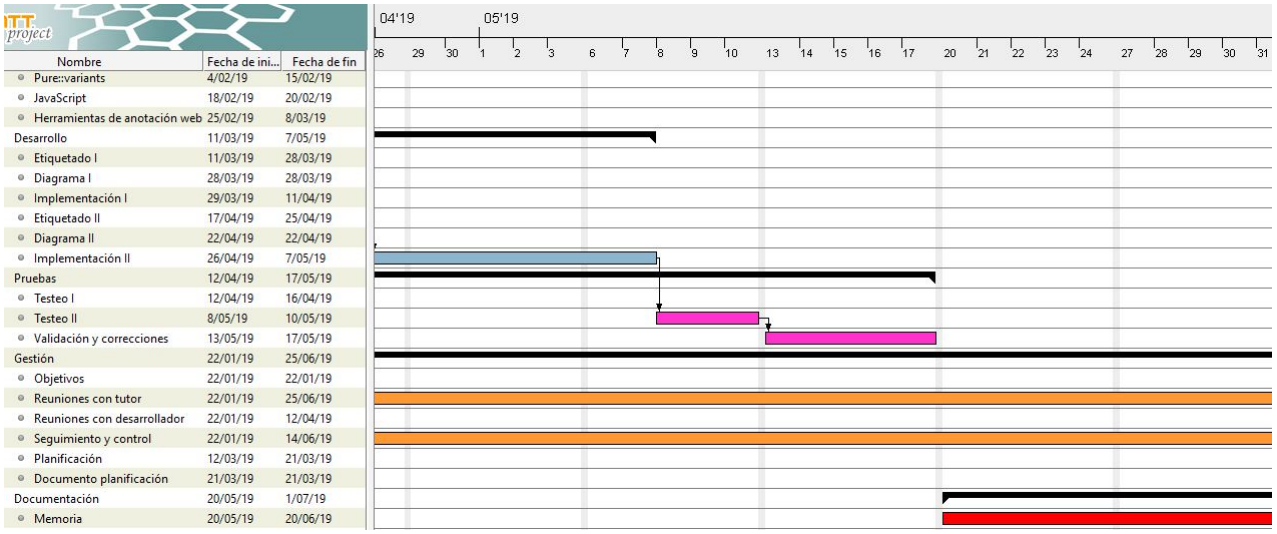


Figura 2.6: Parte del diagrama Gantt (30 de abril - 31 de mayo).

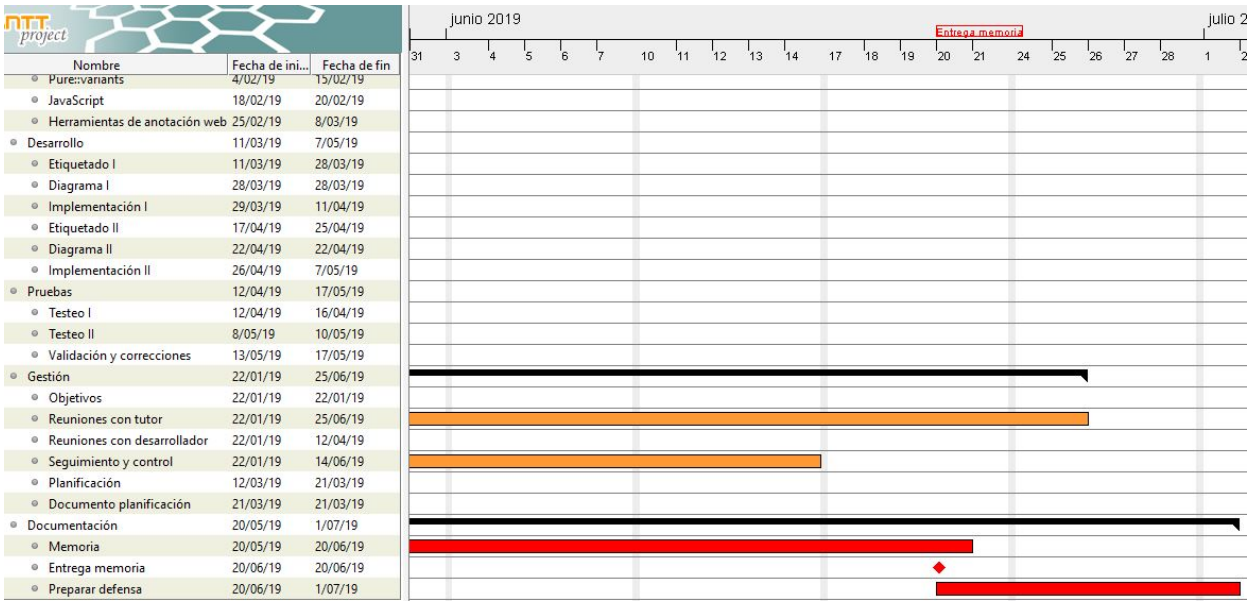


Figura 2.7: Parte del diagrama Gantt (1 de junio - 1 de julio).

2.6. Herramientas

2.6.1. Draw.io

Se trata de una aplicación Web que permite elaborar gráficos y crear diagramas de todo tipo de una manera sencilla. Para el proyecto, se ha utilizado su cliente de escritorio para crear la estructura de descomposición del trabajo y algunas de las figuras de la memoria.

2.6.2. GanttProject

Es un programa de código abierto que se utiliza para planificar y gestionar proyectos utilizando los diagramas de Gantt. Mediante esta herramienta se ha elaborado el diagrama de Gantt para agendar las tareas del proyecto.

2.6.3. GoogleDrive

Es un servicio de alojamiento de archivos en la nube que permite acceder a los archivos desde cualquier lugar gracias al almacenamiento seguro en la nube y a la creación de copias de seguridad. Este servicio se ha utilizado para almacenar documentos relacionados con la planificación y las actas de las reuniones.

2.6.4. DiffMerge

DiffMerge [4] es una aplicación gratuita para comparar código y unir ficheros. Para realizar la comparación del contenido de las carpetas y ficheros de los tres proyectos de partida se necesita utilizar un medio que permita observar las diferencias que hay entre los tres.

DiffMerge permite mostrar gráficamente las diferencias entre dos archivos resaltando en que partes son distintos y permite realizar la edición sobre ellos. Aparte de la comparación a nivel de fichero también permite realizar un Folder Diff, que consiste en realizar una comparación lado a lado de dos carpetas donde se muestra qué archivos están presentes en una u otra carpeta, además, identifica que pares de archivos son idénticos o diferentes.

[4] [SourceGear, 2017]

2.6.5. Review&Go

Review&Go es una de las tres extensiones de navegador que parten como producto de partida para realizar la LPS 5.3. Aparte de ser uno de los productos que hay que analizar para incluir sus *Features* en la LPS, se utilizará para realizar la identificación de *Features* en los ficheros de los proyectos. Para ello, se abrirán los ficheros en el navegador con esta extensión y se anotarán las líneas que se crean correspondientes a una *Feature* con los criterios personalizados que representarán *Features*.

2.6.6. GitHub

GitHub es una plataforma colaborativa para alojar proyectos utilizando el sistema de control de versiones Git. Utilizando Git se consigue llevar el registro de los cambios en archivos y coordinar el trabajo que varias personas realizan sobre ficheros compartidos. GitHub se utiliza con la finalidad de ir guardando continuamente el código de la LPS y los ficheros que pertenecen a la memoria del proyecto.

2.6.7. IntelliJ IDEA

Es un entorno de desarrollo integrado para el desarrollo de aplicaciones software desarrollado y mantenido por la empresa JetBrains. Este entorno es el mismo que se ha utilizado para el desarrollo de los tres productos de partida, así que cuenta con la recomendación del desarrollador. IntelliJ es uno de los entornos más populares para desarrollar en JavaScript y en el proyecto, se utilizará para modificar los ficheros de los productos a la hora de realizar las pruebas necesarias para la implementación de nuevas *Features*.

2.6.8. *Pure::variants*

Es la herramienta elegida para la realización de la LPS. Al tratarse de una aplicación fundamental para la realización del proyecto se le dedica un apartado aparte para presentarla de manera más detallada. 4.2.

2.6.9. TextMaker

Texmaker es un editor de LaTeX gratuito que integra muchas herramientas necesarias para desarrollar documentos con LaTeX en una sola aplicación. Se utilizará para la elaboración de la memoria del Trabajo de Fin de Grado.

2.6.10. Google Docs

Permite crear documentos en línea, editarlos o compartirlos a través de Google Drive. Se utilizará para redactar las actas de las reuniones y compartirlas con el tutor.

2.6.11. Presentaciones de Google

Ofrece la posibilidad de crear presentaciones en línea, modificarlas y compartirlas a través de Google Drive. Se utilizará para crear la presentación del trabajo que se debe presentar delante del tribunal a principios de julio.

2.7. Interesados

Entre los interesados del proyecto se encuentra Xabier Garmendia, el autor del proyecto, que se encargará de realizar el proyecto de una manera planificada para que al finalizarlo se hayan cumplido todos los objetivos con el nivel de calidad adecuado.

El proyecto cuenta con Óscar Díaz como director. Se encargará de guiar al alumno a llevar a cabo el proyecto correctamente y tratará de aconsejarle en los momentos que el alumno requiera ayuda.

Uno de los interesados principales es Haritz Medina, ya que el resultado del proyecto le permitirá tener sus tres productos desarrollados en una LPS que podrá ir aumentándola y extendiéndola durante su tesis doctoral. Haritz estará presente en las decisiones que se tomen sobre la LPS y colaborará en las labores de implementación en caso de necesitar realizar refactorizaciones en el código.

El exalumno de la Facultad de Informática de la UPV, Raúl Medeiros Pérez, conocedor de la herramienta de *pure::variants*, participó en la primera reunión del proyecto para

realizar una breve introducción sobre la aplicación al alumno encargado de realizar la LPS.

Como último interesado, se encuentra el tribunal encargado de evaluar el trabajo y la defensa del proyecto.

2.8. Gestión de riesgos

Existen varios riesgos que podrían poner en problemas el desarrollo del proyecto. Algunos de los riesgos se pueden evitar si se toman medidas de prevención, pero pueden existir otros que no se puedan evitar. Ante los problemas que no se pueden prevenir, es importante tener un plan de respuesta que ayude al proyecto a salir adelante. Los riesgos que se han identificado son los siguientes:

- **Problemas personales:** A lo largo del proyecto, pueden surgir problemas personales inesperados e imprevisibles como accidentes o enfermedades, que pueden causar un retraso en los cumplimientos de los plazos previstos.

  Ante esta situación se tratará de retomar el trabajo lo antes posible y se realizará una nueva planificación si es necesaria.

- **Perdida de información:** La pérdida de información puede resultar un grave problema. Provocaría un retraso en la realización de las tareas porque habría que volver a realizar el contenido que se ha perdido.

  Para evitar que una pérdida de información afecte al desarrollo del proyecto se utilizará el control de versiones en GitHub, donde se almacenará todo el código fuente del proyecto y todos los ficheros de LaTeX para la redacción de la memoria. Los archivos que se gestionan en el ordenador tendrán una copia de seguridad en Google Drive para evitar que se pierdan. Estas medidas también permitirán poder acceder a la información desde cualquier lugar, ya que se encontrará en la nube.

- **Avería del ordenador:** Uno de los problemas más graves que puede ocurrir durante el proyecto es que el ordenador donde se realiza el trabajo sufra una avería y no se pueda hacer uso de él. Esto no afectaría en cuanto a la pérdida de información porque ese problema se cubre en el anterior riesgo identificado. Este problema causaría que no se pueda llevar adelante el proyecto al no poder contar con un ordenador.

Si el ordenador se estropeara, se podría seguir con el proyecto utilizando un ordenador de la Facultad de Informática o si aún queda mucho proyecto por delante, podría plantearse comprar un nuevo equipo.

- **Asignaturas de cuarto curso:** Junto a la realización del proyecto se deben cursar dos asignaturas del cuarto curso, por lo que hasta mediados de abril no se dispondrá de tiempo por las tardes para realizar tareas del TFG. Uno de los retos será compaginar el trabajo que supongan las asignaturas con el TFG.

  Para finalizar el proyecto y el curso de buena manera, hasta finalizar las asignaturas se realizarán las tareas del TFG por las mañanas solo cuando no haya trabajos ni exámenes pendientes. Una vez finalizado el cuarto curso se le podrán dedicar más horas al proyecto.

- **Problemas con la comprensión del código de los tres productos de anotación:** El código de los tres productos de partida es complicado y no contiene información sobre la implementación (solo unas pocas funciones contienen unos comentarios explicando para que sirven). Para una persona que no conozca mucho el lenguaje y la arquitectura de una extensión de navegador puede resultar muy difícil entender el código, por lo que puede causar un gran retraso a la hora de realizar esta tarea.

  En el caso de que suceda este problema se contará con la ayuda de Haritz, que al ser el desarrollador de las extensiones puede explicar y aclarar todas las dudas que surjan.

- **Necesidad de realizar refactorización de código a la hora de implementar:** A la hora de implementar las *Features* en la LPS puede que haya que realizar unos grandes cambios en el código tanto para refactorizar como para cambiar la implementación con el propósito de que se ajuste a la LPS. En el caso de sea necesario cambiar mucho código de la implementación se contará con la colaboración de Haritz, que es capaz de realizar estos cambios rápidamente.

- **Problemas con la licencia de *pure::variants*:** La licencia de prueba de *pure::variants* solo dura un mes, en el caso de que esta termine y no se pueda conseguir otra licencia de prueba nueva, la universidad tratará de conseguir una licencia de estudiante poniéndose en contacto con el soporte de *pure::variants*.

- **Problemas para entender *pure::variants*:** En el caso de encontrar dificultades para entender el funcionamiento de *pure::variants* por insuficiencia de información, se

contactará con Raúl Medeiros para comentar los problemas que hayan surgido y de respuesta de cómo se pueden resolver.

2.9. Sistemas de información y comunicación

Como sistema de información del proyecto se utilizará el ordenador del alumno. En él, se almacenarán todos los documentos que se vayan creando y se necesiten durante el desarrollo, pero también se guardará una copia en Google Drive para disponer de los documentos en la nube. Los documentos se almacenan en una carpeta llamada "TFG" que está dividida en más subcarpetas para tener el contenido más organizado. Por ejemplo, para todos los ficheros relacionados con la planificación se cuenta con una carpeta de nombre "Planificación" dentro de la carpeta principal.

En la carpeta "TFG" hay una carpeta para cada uno de los productos de partida donde se encuentra el código de cada uno y en el proceso de analizar el código para conseguir *Features*, por cada fichero del proyecto, se almacenará un archivo PDF que se obtendrá después de anotarlo con las *Features* identificadas con la extensión *Review&Go*.

La Línea de Productos Software se realizará con *pure::variants*, que se instalará en el ordenador del alumno que realiza el proyecto. La carpeta que contiene el código fuente del proyecto y los ficheros LaTeX para realizar la memoria se irán guardando en un repositorio de GitHub para tener un control de versiones y prevenir posibles pérdidas.

La comunicación entre Xabier Garmendia, Óscar Díaz y Haritz Medina se realizará vía e-mail. Los mensajes servirán para informar al tutor sobre la situación del proyecto, quedar para realizar una reunión, enviarse archivos o resolver dudas. Aun así, para realizar un seguimiento del proyecto se realizarán reuniones entre el alumno y el director.

3. CAPÍTULO

Dominio de anotaciones Web

Para entender el propósito de las extensiones de navegador que se utilizan para la realización del proyecto, y conocer el campo sobre el que se quiere realizar la LPS, resulta necesario conocer varios conceptos y tecnologías que se utilizan en el dominio de las anotaciones Web.

En este capítulo, se recogen las ideas necesarias para conocer qué son las anotaciones Web y su utilidad en las extensiones. Para gestionar las anotaciones es necesario especificar cómo se estructuran las anotaciones y para ello, se utiliza el estándar W3C para anotaciones Web. Esta terminología será parte importante para la definición de las anotaciones que se quieran utilizar a la hora de crear el producto partiendo del Modelo de Características de la LPS. Por último, se presenta *Hypothes.is*, un proyecto involucrado en las anotaciones Web y un elemento importante de las extensiones de anotación Web del proyecto.

3.1. Anotaciones Web

Las anotaciones Web son anotaciones *online* que se realizan externamente sobre un recurso web, como por ejemplo una página web o un documento. Por medio de una anotación, el usuario puede agregar un contenido donde se encuentra información al recurso web, sin tener que modificarlo mediante edición. Las anotaciones se realizan sobre el recurso web, pero no afectan al recurso porque se distinguen dos capas diferentes: la capa del recurso existente y la capa de la anotación. [Wikipedia, 2019c]

La capa de la anotación es visible para el anotador y para los usuarios que disponen del mismo sistema de anotación. El sistema de anotación es un tipo de herramienta software que crea esta segunda capa encima del recurso. Actualmente, los navegadores no contienen esta segunda capa integrada y por ello, se debe contar con ciertas extensiones web para dichos navegadores que se encarguen de proporcionarla. Entre las extensiones conocidas se encuentran *Hypothes.is*, *zipBoard* o *Diigo*.

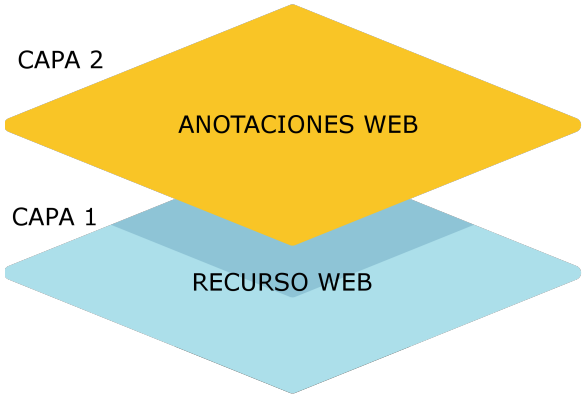


Figura 3.1: Ilustración de la idea de la capa de anotación

Una anotación Web puede ser realizada para fines como calificar un recurso web, mejorar o adaptar el contenido, discutir los contenidos del recurso de forma colaborativa, añadir explicaciones, etcétera.

3.2. Estándar W3C de anotaciones web

El 23 de febrero de 2017, las anotaciones Web dieron un gran paso cuando los estándares de W3C [1] publicaron tres nuevas recomendaciones,[2] creando así anotaciones estandarizadas para habilitar las anotaciones en la Web y poder compartirlas entre sistemas.

La estandarización resulta clave para que los sistemas de anotación puedan realizar las anotaciones de manera común y poder compartirlas entre diferentes hardware y plataformas software. La interoperabilidad entre sistemas de anotación es uno de los grandes beneficios que aporta el estándar. Si las anotaciones se realizan de esta manera no estarán ligadas a las herramientas que las diseña, sino que serán de acceso abierto a cualquier

[1] World Wide Web Consortium. Es el organismo de estándares para la web que tiene como finalidad la creación de una Web universal.

[2] Popularmente las recomendaciones del W3C son conocidas como estándares Web.

otra herramienta y promoverían la portabilidad. Las anotaciones de W3C son recursos web, por lo tanto, son direccionables (son un recurso localizable en la Web a través de una dirección). Al convertir las anotaciones en recursos web, se consigue que puedan ser localizadas, reutilizadas y complementadas por terceros.

Los tres estándares se presentan a continuación:

- Web Annotation Data Model [W3C, 2017a]: Define la estructura de las anotaciones. Se describe el modelo de datos y el formato de la anotación.
- Web Annotation Vocabulary [W3C, 2017c]: Especifica la semántica establecida que es utilizada por el modelo de datos.
- Web Annotation Protocol [W3C, 2017b]: Describe los mecanismos de transporte para la creación, gestión y recuperación de anotaciones siguiendo los dos anteriores estándares.

La recomendación del modelo de datos es interesante porque establece cómo se debe definir la estructura de la anotación y además la especificación proporciona un formato de JSON para facilitar la creación y el consumo de anotaciones basadas en el modelo y adaptada al vocabulario.

Según el modelo de datos una anotación se considera un conjunto de recursos conectados que normalmente incluye un cuerpo (*body*) y un destino (*target*), donde el *body* está relacionado con el *target*. Esta definición de modelo de la anotación da como resultado un modelo con tres partes como se puede observar en la figura 3.2.

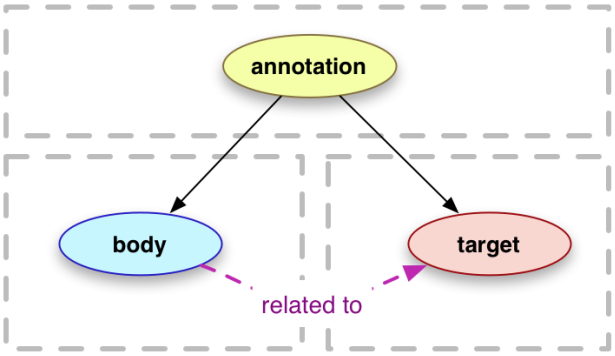


Figura 3.2: Modelo de datos de una anotación web.

En el *target* que contiene la anotación se guarda la información sobre el recurso web anotado y, por otro lado, en el *body* se guarda el contenido de la anotación, como por ejemplo un comentario sobre el recurso web que se ha anotado.

Una anotación no se hace necesariamente sobre todo un recurso web (un documento, vídeo u hoja de cálculo), si no que es una práctica común hacerlo sobre un fragmento de ese recurso (ya sea un párrafo, un fragmento de un vídeo o una celda de una hoja de cálculo). La selección del recurso web que se quiere anotar se realiza por medio de un término proporcionado por Web Annotation Vocabulary llamado *selector*.

Hay múltiples tipos de selectores, [3] con el objetivo de hacer referencia a los fragmentos anotados e incluso se utilizan una combinación de ellos. Por ejemplo, a la hora de anotar texto el selector bien puede funcionar guardando la posición de la primera y última palabra del texto anotado (*TextPositionSelector*) o guardando el texto que precede y prosigue de la anotación realizada (TextQuoteSelector). En el caso de que el recurso sea un vídeo, el selector podría ser el tiempo inicial y final del trozo de vídeo que se quiere anotar.

W3C proporciona más propiedades para añadir a la anotación como por ejemplo reconocer quien es su autor (*creator*), su fecha de creación (*created*), el objetivo con el que se realiza (*motivatedBy*).

En la figura 3.3 se muestra una instancia de una anotación que sigue el modelo del W3C. La anotación (anno1) como modelo de datos contiene un *body* y un *target*.

En el *body* se encuentra el contenido de la anotación: “myComment”. En este caso el valor con el que se anota se trata de un texto que tiene el objetivo de comentar la anotación tal y como se indica a través de la propiedad *motivatedBy*.

En el *target* se encuentra la información acerca del recurso que se anota. La anotación se realiza sobre el recurso “webpage1” y el fragmento anotado se trata de “annotation” que para localizarlo dentro del recurso se utiliza un selector del tipo TextQuoteSelector que guarda el texto que precede a la anotación (“this is an”) y el texto que lo prosigue (“that has”).

A parte del *body* y el *target* también se guarda más información mediante propiedades. Las propiedades que se almacenan son *motivatedBy*, *creator* y *created* que guardan el propósito (comentar), el autor (person1) y la fecha de creación (2018-12-18) de la anotación respectivamente.

[3] Tipos de selectores

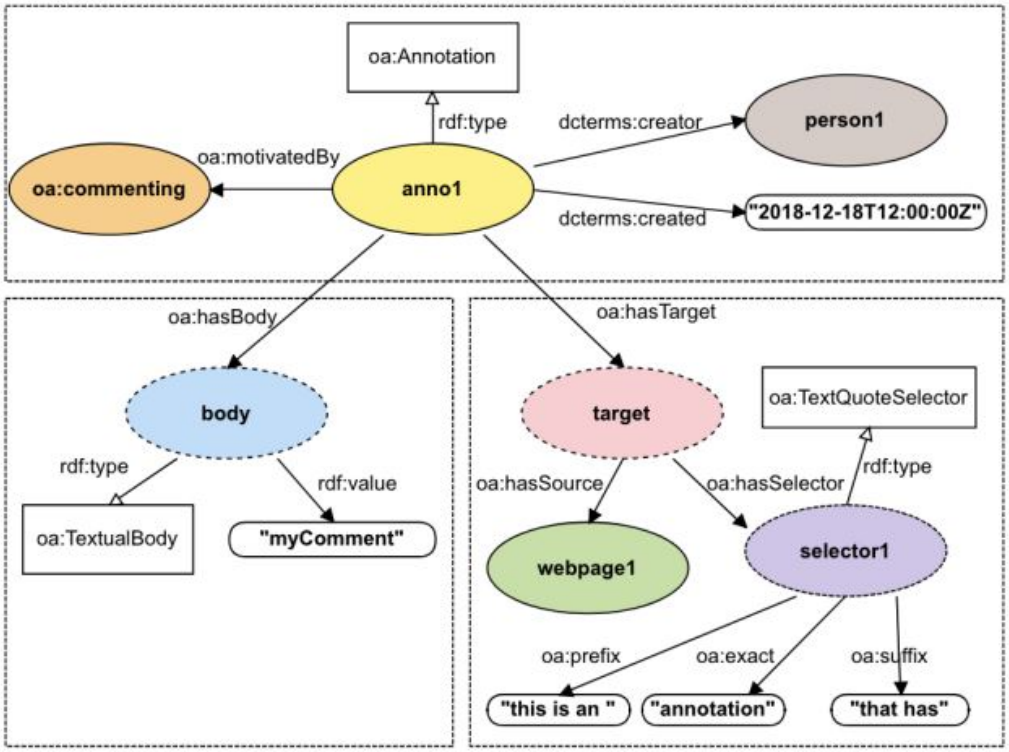


Figura 3.3: Ejemplo de una anotación web siguiendo el estándar W3C.

Esta información sobre las anotaciones resulta importante para entender cómo se compone una anotación y comprender mejor como se definen en las extensiones del proyecto.

3.3. Hypothes.is

Hypothes.is[4] es un proyecto software de código abierto dirigido por Dan Whaley cuyo fin es crear una plataforma en la que los usuarios puedan realizar anotaciones Web y puedan ser compartidas. El proyecto tuvo la financiación de *Kickstarter*[5] y es una organización sin ánimo de lucro que cuenta con la ayuda de las fundaciones Knight, Mellon, Shuttleworth, Sloan, Helmsley y Omidyar.

Hypothes.is se basa en el estándar para anotar documentos desarrollado por Open Annotation Collaboration, un consorcio que incluye a Internet Archive, NISO, O'Reilly Books, Amazon, Barnes and Noble y varias instituciones académicas.

[4]La información acerca de *Hypothes.is* se ha extraído de [Wikipedia, 2019a] y [Hypothes.is, 2019b]
[5]Es un sitio web que facilita la captación de recursos monetaria para proyectos creativos.

La organización se asocia con desarrolladores, editores, investigadores e instituciones académicas para desarrollar una plataforma para la próxima generación de aplicaciones web de lectura y escritura.

Hypothes.is es un proyecto que cada vez va siendo más conocido y utilizado. Actualmente, cuenta con más de 215.000 usuarios y ya contiene más de 5 millones de anotaciones. [Hypothes.is, 2019a]

Figura 3.4: Realización de anotaciones con la extensión de *Hypothes.is*

Para realizar las anotaciones disponen de una extensión para *Google Chrome*, pero también disponen de una página web donde se pueden consultar las anotaciones y que proporciona la opción de crear y gestionar diferentes grupos de anotaciones para poder compartirlas entre los usuarios.

En la 3.5 se muestra la interfaz de la página web de *Hypothes.is*. En ella se pueden consultar los grupos y realizar todo tipo de búsquedas sobre las anotaciones realizadas (por grupo, tag, contenido anotado,...). En la parte izquierda de la figura se pueden observar las anotaciones realizadas por el usuario y en la parte derecha, en la pestaña de "Grupos", se ven los grupos de anotaciones que tiene.

Figura 3.5: Interfaz de *Hypothes.is* donde ver los grupos y anotaciones.

El objetivo de este capítulo ha sido cubrir los conocimientos requeridos para conocer el dominio sobre el que se realiza la LPS. Para ello, es importante haber aprendido para que sirven las anotaciones Web y que tipo de operaciones se pueden hacer con ellas para poder entender porqué y cómo se utilizan en los productos de partida. Las anotaciones Web están estandarizadas por W3C, lo que permite definir las anotaciones de una manera global ya preestablecida.

En los productos que se basa la LPS desarrollada, *Hypothes.is* se utiliza con el fin de almacenar las anotaciones. Para ello, se hace uso de la API que proporciona, y mediante llamadas permite realizar acciones como crear un grupo para anotaciones, crear anotaciones, recuperar anotaciones, etcétera.

4. CAPÍTULO

Líneas de Productos Software y *pure::variants*

El objetivo de este capítulo es explicar la arquitectura de las Líneas de Productos Software. Para ello, en primer lugar, se situará el contexto en el que se deben utilizar. Seguidamente, se resaltarán los beneficios más importantes que se adquieren mediante este enfoque de producción que otros no proporcionan. Además, se comentan algunas de las técnicas y procesos de desarrollo de Líneas de Productos Software. Finalmente, se presenta la herramienta escogida para la realización de la LPS del proyecto y algunos de los conceptos a tener en cuenta para el desarrollo.

4.1. Líneas de Productos Software

Hasta hace un tiempo, cuando una organización que crea múltiples aplicaciones software similares tenía que crear un nuevo producto para un cliente, la manera con la que tenía que afrontarlo era mediante la elaboración de un nuevo proyecto partiendo desde cero.

Actualmente, la opción más habitual para lidiar con este tipo de problema y no tener que empezar un proyecto desde cero es realizar una copia del producto disponible y realizar las modificaciones que sean necesarias sobre el nuevo proyecto para que se ajuste a las pretensiones del cliente. De realizar el nuevo producto de esta manera, la labor necesaria para terminar su elaboración sería implementar las nuevas variantes que requiere el producto, y realizar los cambios necesarios sobre el código copiado para amoldarlo a las nuevas necesidades.

Producir los nuevos productos usando la metodología *clone&own* en un principio puede resultar interesante porque agiliza el desarrollo, pero hay que tener en cuenta que a partir de la nueva copia se crea un producto más para mantener. A pesar de que el nuevo producto tenga la misma estructura y tenga gran parte del código igual, tendrá que evolucionar y mantenerse de manera independiente, sin poder aprovechar las ventajas que podrían obtenerse de la semejanza entre productos. Una organización que siga utilizando este proceso de producción se va encontrar en una situación en la que le va resultar muy caro y complicado poder mantener todas estas aplicaciones.

Figura 4.1: Producción siguiendo el método *clone&own*.

En la figura 4.1 se muestra la idea que sigue el sistema de producción *clone&own*. Partiendo del producto número uno se crea el segundo producto, que tiene gran parte del código del producto anterior y otra parte que es el código nuevo. Del segundo producto se crea luego el tercer producto que contiene partes de los otros dos productos anteriores y código nuevo de las nuevas variaciones. Del último producto se crearán unos nuevos, y así sucesivamente. Si en algún momento fuese necesario cambiar parte del código azul en los productos, habría que localizar en que productos se encuentra y aplicar la modificación en cada uno de ellos.

Supongamos que una empresa que mantiene una familia de productos software que cuenta con diez productos debe realizar un cambio en una parte del código que es común en cada aplicación. Tener que realizar el cambio supondría tener que modificar líneas de código una por una en cada uno de los proyectos, por lo tanto, sería una labor tediosa, pesada y repetitiva para los desarrolladores o los encargados de mantener el software. Además, las peticiones de los clientes están a la orden del día, y en cuanto la organización menos se lo espere, se puede encontrar en una situación en la que tiene que mantener un gran número de productos. ¿Sería posible tener que mantener más de cien productos y tener que realizar el mismo cambio en cada uno cuando la situación lo requiera? Esta claro que posible sería, pero el coste que supone es extremadamente elevado. Por esta razón, la organización debe buscar la manera más eficiente y económica que le ayude a gestionar

y aplicar adecuadamente el conjunto común de activos reutilizables (*core assets*) que mantiene.

La producción de productos personalizados en serie no es un problema exclusivo del software. Esta situación también se da en casos como en la fabricación de otros productos como vehículos, ordenadores u otro grupo de productos que puedan requerir diferentes variaciones y formar una familia de productos. Por ejemplo, en un sistema de producción de electrodomésticos, la personalización en serie se consigue por medio de una cadena de montaje o línea de producto. Las líneas de producto delimitan las variantes de sus productos a un conjunto preestablecido, y optimizan los procesos para estas variantes.

Ante este importante reto de cómo enfocar la creación de nuevos productos, surgió la idea de las Líneas de Productos Software, cuyo fin es desarrollar y mantener familias de productos sacando partido a los elementos comunes y gestionando de manera eficaz las variaciones.

La definición utilizada por el Instituto de Ingeniería de Software es la siguiente: "*Las Líneas de Productos Software son un conjunto de sistemas software, que comparten un conjunto común de características que satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un conjunto común de activos reutilizables de una manera preestablecida*". Una producción basada en una LPS trata de gestionar los componentes comunes y las variaciones a soportar que existen entre los productos en un único espacio.

La reutilización que se consigue mediante la técnica del *clone&own* es oportunista, ya que generalmente se utilizan componentes de software del anterior producto, que pueden no ser necesarios en el nuevo. En una Línea de Productos, en cambio, la incorporación de nuevas variantes se realiza de forma sistemática y controlada, por lo que el producto generado solo tendrá los componentes reutilizables si los resulta necesarios, obteniendo así una calidad de precisión a las peticiones del cliente.

El objetivo de la figura 4.2 es realizar un parentesco para entender la idea de la LPS. En la figura 4.1 los productos se crean a partir del producto anterior. Con este nuevo enfoque, todos los productos se crean directamente desde la línea de producto. En ella se recogen todos los elementos comunes y las distintas variabilidades preestablecidas hasta el momento, con las que realizando distintas combinaciones, se pueden producir diferentes productos. La pintura azul representa el conjunto común de características aplicable en todos los productos que dispone la línea de productos y los otros dos colores, el rojo y el verde, representan las necesidades específicas para ciertos productos.

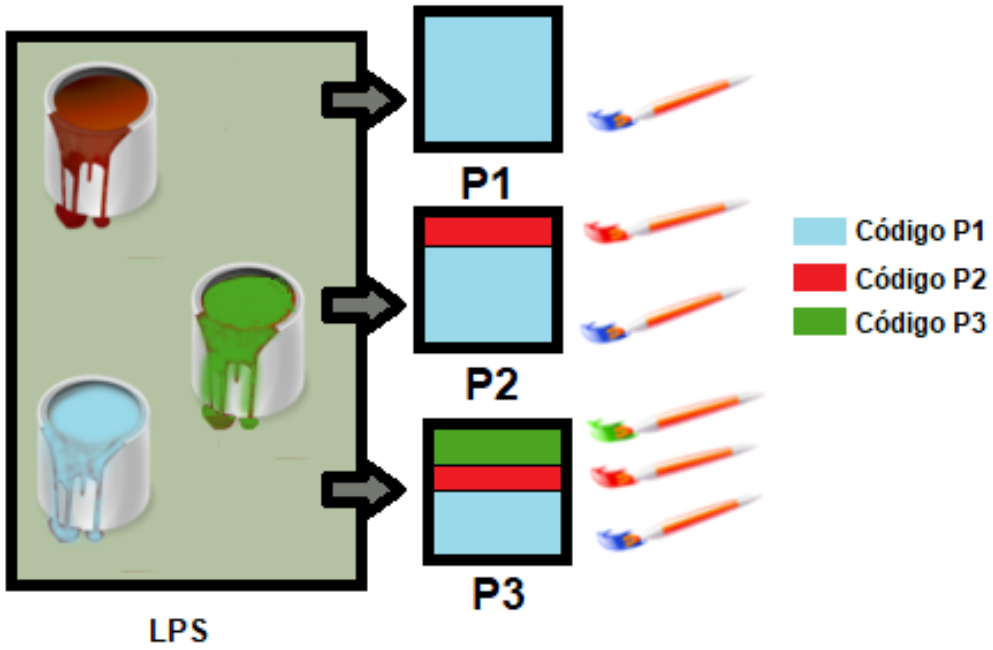


Figura 4.2: Producción siguiendo la arquitectura de Línea de Producto.

4.1.1. Beneficios

Utilizando la arquitectura LPS ya no hace falta realizar el mantenimiento de cada producto por separado. Ahora todos los productos evolucionan y se mantienen a la vez, por lo que los ingenieros solo tienen que centrar en un único punto. Los esfuerzos de mantenimiento requeridos se realizan en la LPS, y los cambios se diseminan entre todos los productos que tienen parte de ese código modificado. Por lo tanto, al detectar un error en uno de los productos ese mismo error se corregiría en todos los productos de la LPS, reduciendo así la tasa de defectos de los productos. Ir mejorando y arreglando constantemente los elementos comunes (*core assets*) hace que a lo largo del tiempo estén ya probados, y que los nuevos productos que se creen a partir de la LPS difícilmente fallen en estos puntos.

Hay muchas razones por las que se aconseja seguir esta línea. Cuando un producto entra en un mercado internacional es posible que tenga adaptarse a diferentes leyes y culturas. Problemas como el idioma requieren que el producto cuente con características específicas de ese país y, no obstante, no es una tarea sencilla ir creando software para cada cliente, por lo que una LPS para ese producto sería una buena solución.

La arquitectura LPS genera unos grandes beneficios en productividad y costes. Con esta nueva línea de producir productos los ingenieros van a ver incrementada su productividad,

ya que no van a tener que aplicar tanto esfuerzo en el mantenimiento y en la puesta en marcha de los productos. Ahora los ingenieros de software no se centran en la creación de un nuevo producto entero, sino que deben desarrollar nuevas variaciones para hacer crecer el grado de variabilidad de la LPS y poder generar nuevos productos.

Tal y como se ha mencionado anteriormente, los costes que se generan a partir de la producción de productos siguiendo el método *clone&own* son extremadamente grandes. Cuanto más mayor es el número de productos, el esfuerzo y coste requerido para la elaboración y el mantenimiento crece. Esta línea requiere que haya un equipo humano detrás de cada producto, y de esta manera, el número de productos que se pueden gestionar eficazmente es escaso.

Con un entorno de líneas de producto, los grandes costes de mantenimiento desaparecen porque ahora se pueden aplicar los cambios a todos los productos desde la misma LPS, lo que permite gestionar y mantener un mayor número de productos.

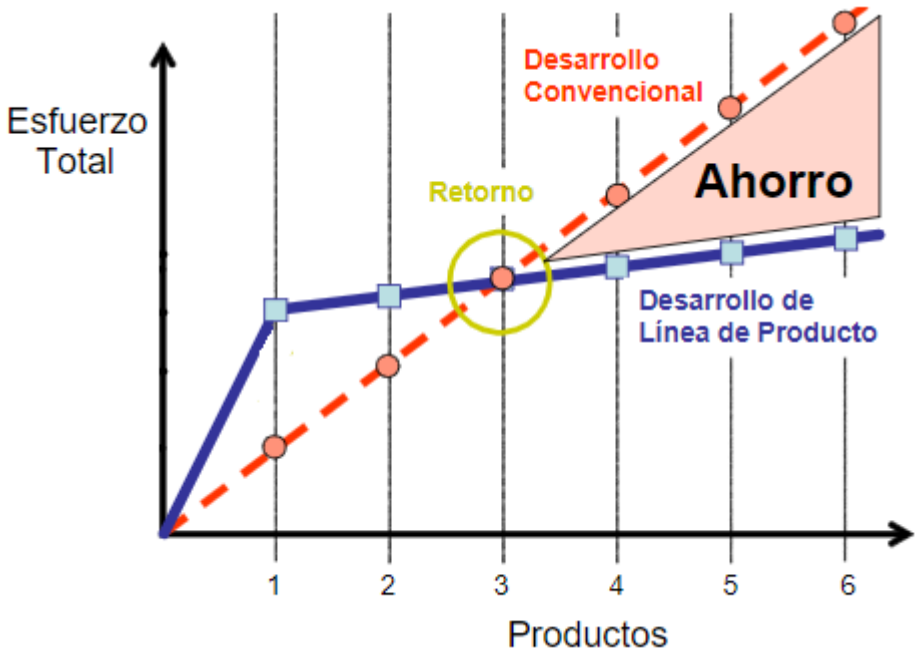


Figura 4.3: Comparación de costes: Producción convencional vs LPS.

Mediante la figura 4.3 [1] se muestra la comparación del coste en esfuerzo que requiere el desarrollo y mantenimiento de una familia de productos utilizando el desarrollo convencional (*clown&own*) y el enfoque de LPS.

[1] El gráfico se ha extraído de la siguiente lectura [Diaz and Trujillo, 2010]

Con el desarrollo tradicional no se obtienen ahorros importantes de esfuerzo. Al ser productos derivados de otros que evolucionan de manera independiente, el esfuerzo de mantenimiento que necesita el producto anterior lo va requerir el nuevo producto, a lo que se le suma el coste que conlleve su personalización.

El enfoque de LPS es contrario al convencional. Al principio, necesita realizarse un estudio del dominio para determinar los elementos comunes y las variaciones a soportar para desarrollar la plataforma de la LPS. Dentro de este proceso entran las actividades relacionadas con la Ingeniería de Dominio (véase 4.1.3). El gran coste que supone la LPS se realiza inicialmente incluso antes de realizar el primer producto que además dependerá del proceso que se elija para desarrollar la LPS (véase 4.1.2). Con la LPS, el esfuerzo inicial se verá recompensado a la hora de comenzar la generación de productos, ya que de un producto a otro el esfuerzo es casi insignificante. Si el desarrollo se ha realizado de manera adecuada, se estima que para la realización del tercer producto la arquitectura LPS es rentable y los productos producidos a partir de este punto requerirán menos esfuerzo siguiendo este enfoque.

Del párrafo anterior se puede concluir que una LPS será rentable cuando se estima que la familia de productos será numerosa. Si la familia se limita a tres productos, cabe la opción de que el desarrollo convencional sea menos costoso, pero no permitirá gran opción de seguir creciendo.

4.1.2. Estrategias para iniciar una LPS

Como bien se ha mencionado anteriormente, el mayor esfuerzo que requiere la LPS se realiza en el proceso de desarrollo de la plataforma.[2] El modelo de proceso para desarrollar que se utilice condicionará el desembolso inicial y el esfuerzo requerido para poner en marcha la LPS. Hay tres modelos diferentes de iniciar el desarrollo de la LPS: modelo proactivo, reactivo y extractivo.

Realizar el desarrollo de una manera proactiva es la que más coste supondría inicialmente. El modelo proactivo se basa en realizar un estudio del dominio para abarcar todos los productos posibles en los siguientes años buscando un enfoque a largo plazo.

Si el reto de planificación a largo plazo resulta difícil de proceder, existe la opción de realizar un desarrollo reactivo. Con este proceso de desarrollo se realiza una planificación que dé cobertura a un corto plazo y la idea es ir introduciendo las nuevas *Features* e ir

[2] La información de este apartado se ha extraído de la lectura [Diaz and Trujillo, 2010]

acometiendo cambios según se vayan solicitando por los clientes o las necesidades del mercado.

Por último, está el modelo de desarrollo extractivo. Este proceso de desarrollo resulta ser el más importante para el contexto de este proyecto, porque es el que se va seguir para el desarrollo de la plataforma de la Línea de Productos del dominio de anotaciones Web. Cuando una organización dispone ya de varios productos que les resulta difícil de mantener, se empiezan a plantear la idea de pasar los productos a una LPS. En estos casos se utiliza el enfoque extractivo, que en vez de comenzar la LPS desde cero, comienza a desarrollar la plataforma partiendo de unos productos de la familia ya existentes.

El desarrollo de la LPS se puede distinguir en dos procesos de ingeniería diferentes: Ingeniería de Dominio y Ingeniería de Producto.

4.1.3. Ingeniería de Dominio

Mediante la Ingenieria de Dominio se desarrolla la infraestructura de la LPS. Para ello, se requiere realizar un estudio del dominio, definir el alcance de la LPS y definir las *Features* para realizar la implementación de los *core assets* reutilizables.

Al realizar un análisis del dominio se recogen los requisitos mediante *Features*. Una vez obtenidas las *Features* hay que especificar la variabilidad que tendrán estableciendo si son obligatorias u opcionales. El conjunto de *Features* y su variabilidad se plasman en un Modelo de Características (*Feature Model*), que será el resultado del análisis del dominio. En el *Feature Model* se establecen las relaciones que hay entre las *Features* de la LPS.

Tras obtener el diseño de la LPS, se deben implementar las funcionalidades de las *Features* en el proyecto de la LPS, para ello, en este trabajo se va contar con la herramienta *pure::variants*, que mediante mecanismos de etiquetas establece que partes del código pertenece a cada *Feature*. Para establecer que ficheros son los que formarán el proyecto, es necesario mapear esos archivos con un *Family model* 4.2.2.

Los conceptos de *Feature Model* 4.2.1, *Family Model* 4.2.2 y el mecanismo de etiquetado 4.2.3 se detallan en el apartado de *pure::variants*.

4.1.4. Ingeniería de Producto

La Ingeniería de Producto aprovechando la infraestructura que proporciona la Ingeniería de Dominio, desarrolla los productos concretos para los clientes. En el estudio de los

requisitos del cliente se producirá un análisis de los posibles elementos que se pueden reutilizar y se creará un diseño específico eligiendo en la configuración del producto los elementos comunes y variables que se necesiten. Después de realizar la configuración, se realiza la transformación del producto resolviendo los puntos de variación. En el caso de *pure::variants* la configuración del nuevo productos se realiza por medio de modelos VDM [3] y en la transformación se analizan las restricciones establecidas y se mapean los ficheros con el *Family Model* para crear el producto.

4.2. Pure::variants

Pure::variants es una aplicación basada en Eclipse para gestionar las variantes del desarrollo de software mediante la arquitectura de LPS. Por medio de diferentes herramientas, da soporte a cada fase del proceso de desarrollo para finalmente crear soluciones válidas automáticamente de las *Features* elegidas.[4]

A la hora de crear la infraestructura de la LPS, el análisis y diseño del dominio se realizan reuniendo el conjunto de *Features* en un Modelo de Características (*Feature Model*). Para la implementación del dominio de la familia software se utilizan los *Family Models*, que describe la arquitectura de la familia y lo conecta al *Feature Model* con las reglas apropiadas. La creación de variantes de productos se realiza transformando un VDM que contiene las *Features* seleccionadas para la generación de ese producto.

4.2.1. Feature Model

El Modelo de Características (*Feature Model*) es un modelo que ayuda a organizar el conjunto de *Features* que contiene la LPS y sirve para capturar la variabilidad de cada una. Los *Feature Models* tienen la estructura jerárquica de un árbol donde cada nodo representa una *Feature* de la LPS. La variabilidad se representa mediante la ilustración de diferentes arcos entre los nodos. En un *Feature Model* se pueden encontrar cuatro tipos de *Features*:

[3] *Variant Description Model*

[4] La información acerca del funcionamiento y las características de esta herramienta se ha extraído de la [pure systems, 2019b]

- **Obligatoria (*mandatory*)**: Son las *Features* que son obligatorias en todos los productos que se generan en la LPS.
- **Opcional (*optional*)**: Son *Features* que no son obligatorias y se pueden elegir a la hora de crear el producto. Aportan variabilidad a la LPS porque son *Features* que no aparecen en todos los productos.

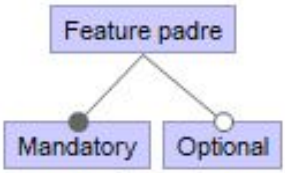


Figura 4.4: *Mandatory Feature* y *Optional Feature*.

- **Disyunción exclusiva (*alternative*)**: Es un grupo de *Features* donde solamente puede ser seleccionada una de ellas.

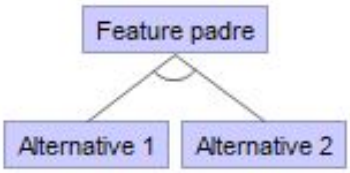


Figura 4.5: *Alternative Features*.

En la figura 4.5 se muestra cómo se reflejan las *Feautures* de disyunción exclusiva en el diagrama. El conjunto de *Feautures* sobre el que hay que elegir se indica dibujando un arco entre las posibles opciones.

- **Disyunción inclusiva (*or*)**: Es un grupo de *Features* donde al menos una de ellas debe ser seleccionada.

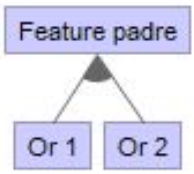


Figura 4.6: *Or Features*.

En la figura 4.6 se ilustra como reflejar las *Feautures* de disyunción inclusiva en el diagrama. El conjunto de *Features* que se puede elegir se indica dibujando un arco relleno entre las posibles opciones tal y como aparece en la figura.

Con las diferentes formas de reflejar las *Feature* se forma el *Feature Model*. En la figura 4.7 se puede observar un ejemplo de un *Feature Model* utilizando todos los tipos de *Features* contemplados anteriormente para representar una aplicación de mensajería.

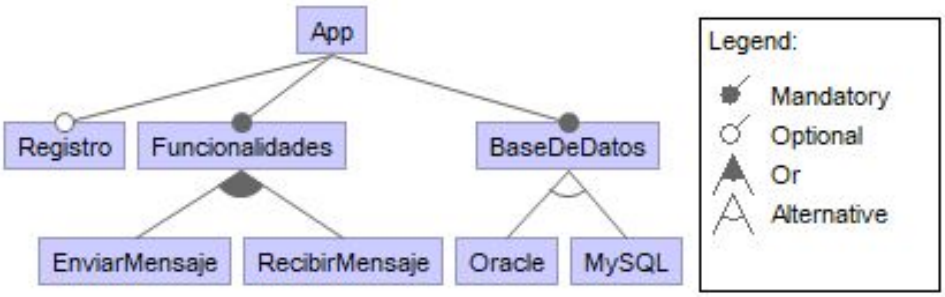


Figura 4.7: Ejemplo de un *Feature Model*.

Según el diagrama, en algunos de los productos existe la opción de registrarse en la aplicación por medio de la *Feature* "Registro". Todas las aplicaciones generadas a partir de este *Feature Model* tienen por obligación mínimo una funcionalidad. Un usuario de la aplicación puede enviar un mensaje, recibir un mensaje o realizar ambas acciones. Por último, se utiliza una base de datos para gestionar los datos de la aplicación y para ello se cuenta con dos alternativas de sistemas de gestión diferentes de las que habrá que elegir una: Oracle o MySQL.

En la figura 4.8 se ilustra el mismo *Feature Model* de antes, pero esta vez representado con las notaciones de *pure::variants*. El diagrama se realiza en un fichero con extension

Figura 4.8: Ejemplo de un *Feature Model* en *pure::variants*.

.xfm y para componer el diagrama se van añadiendo *Features* sobre las *Feature* padre especificando por cada una que tipo de variabilidad tendrá.

El ejemplo anterior es un caso que podría ser real. Pero ¿tiene sentido que un usuario pueda recibir mensajes sin estar registrado? En este caso no tendría ningún sentido y la solución para ello es establecer restricciones en el *Feature Model*.

En *pure::variants* se dispone de muchos tipos de restricciones para establecer entre las *Features*, pero en este proyecto solo se contemplan dos de ellas:

- **X Requires Y**: La *Feature* X requiere que la *Feature* Y este seleccionada.
- **X Conflicts Y**: La *Feature* X requiere que la *Feature* Y no este seleccionada.

Aplicando estas posibles restricciones, para solucionar el problema planteado para el diagrama 4.7 habría que establecer la siguiente restricción: “RecibirMensaje” Requires “Registro” y en *pure::variants* se plasmaría de la manera que se refleja en la figura 4.9

Figura 4.9: Ejemplo de un *Feature Model* en *pure::variants* incluyendo una restricción.

4.2.2. Family Model

El *Family Model* es el modelo encargado de relacionar los ficheros físicos que se utilizan para la generación de los productos con las *Features* de la LPS. El modelo describe toda la arquitectura de la LPS porque contiene donde se localizan todos los ficheros que se pueden encontrar en los productos ya que gestiona todo el código del proyecto de la LPS.

A la hora de transformar un modelo a un producto, el *Family Model* se encarga de hacer la selección de los ficheros que formarán el producto. Para ello, realiza una copia de los ficheros que sean solamente necesarios en el *output* del nuevo producto y elimina las líneas de código en el caso de ser ficheros que sigan mecanismos de etiquetación 4.2.3.

Un *Family Model* sigue una estructura jerárquica que contiene tres tipos de elementos y no tiene un límite de profundidad. Un elemento solo se incluye en el resultado final cuando su padre está incluido y ninguna restricción le afecta.

Los elementos que componen la estructura del *Family Model* son los siguientes:

- **Componentes (*Components*)**: Son entidades que sirven para agrupar elementos que se quieren representar en el *Family Model*. Por ejemplo, para representar las carpetas del proyecto.
- **Partes (*Parts*)**: Hace la relación entre un componente y los *source elements*. Una parte, es una representación lógica de los ficheros. Puede ser un elemento de un lenguaje de programación, como una clase o un objeto. Dentro de una parte hay uno o varios *source elements*.
- **Elementos fuente (*Source elements*)**: Son las representaciones físicas de las partes. Cada *source element* representa un fichero real del código de la LPS. El tipo del *source element* determina como se debe tratar el código del fichero a la hora de generarlo en un nuevo producto. *Pure::variants* soporta varios tipos, pero en el proyecto solo se contempla el uso de tres:
  - **ps:file**: A la hora de generar el producto, el fichero se copia tal y como es en el destino especificado.
  - **ps:pvscltext**: Este tipo se debe especificar para los ficheros que en su código tengan partes que sean de diferentes *Features*. Para que se genere correctamente con el código necesario se usan etiquetas *ifdefs* que se evalúan a la hora de transformar el modelo.

- **ps:pvsclxml**: Sigue la idea del tipo anterior, pero esta vez se utiliza en los ficheros de lenguaje de marcado. A través de un atributo de las etiquetas, se especifica a que *Feature* pertenece cada elemento del fichero.

En el apartado 4.2.3 se explica más en detalle el uso de los últimos dos tipos.

A los elementos que forman la estructura del *Family Model* también se les puede añadir restricciones por medio del lenguaje pvSCL [5] para asociar los ficheros a ciertas *Features*. Por ejemplo, se puede hacer que el fichero "registro.js" solo aparezca en el producto final cuando la *Feature* "Registro" este seleccionada.

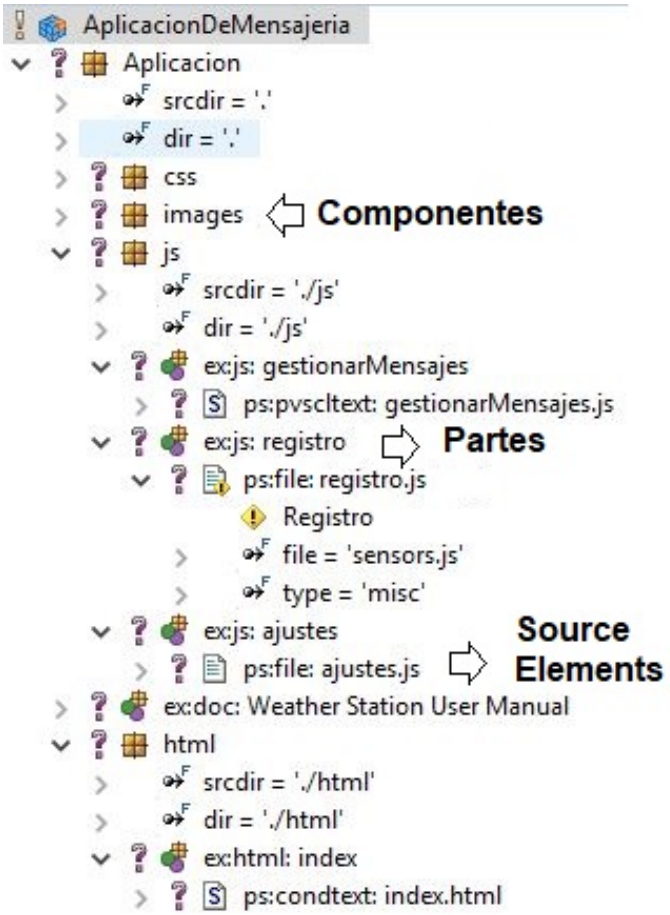


Figura 4.10: Ejemplo de un *Family Model* en *pure::variants*.

En *pure::variants*, un *Family Model* se representa mediante un fichero .ccfm y se forma siguiendo la estructura de la figura 4.10, en la cual se refleja lo que podría ser un *Family*

[5] *pure::variants Simple Constraint Language.*

Model para la aplicación de mensajería. Los componentes formarían las carpetas del proyecto donde se encuentra todo el código. Dentro de los componentes se encuentran las partes (*parts*) que guardan los *source elements*. En la imagen se pueden ver los tres tipos de *source elements* que se han mencionado anteriormente.

El fichero "gestionarMensajes.js" es del tipo *ps:pvscltext* porque puede tener código tanto de la *Feature* "EnviarMensaje" como de la "RecibirMensaje" y no utiliza el lenguaje de marcado.

El fichero "registro.js" tiene una restricción sobre la *Feature* "Registro" por lo que solo aparecerá en el resultado final cuando la *Feature* esté seleccionada.

El archivo "ajustes.js" es del tipo *ps:file*, por lo tanto, este fichero se copiará en todos los productos que se creen usando este *Family Model*.

Finalmente, entre los ficheros HTML nos encontramos el archivo "index.html" del tipo *ps:pvsclxml* porque tiene etiquetas del lenguaje de marcas que solo serán necesarias para ciertas *Features*.

Mediante los atributos de los componentes *srcdir* y *dir*, se establece cual es el *path* de origen y destino del fichero respectivamente.

4.2.3. Etiquetación de ficheros

El objetivo de este apartado es explicar cómo se etiquetan los ficheros que tienen código que pertenece a diferentes *Features*. Para ello hay dos mecanismos posibles: uno sencillo para los ficheros de lenguaje de marcado y otro a través de etiquetas *ifdefs* para el resto de ficheros.

Etiquetación de ficheros de lenguaje de marcado

Este mecanismo de etiquetación es exclusivo para ficheros de lenguaje de marcas como HTML o XML. Mediante atributos especiales en las marcas del documento se especifica a que *Feature* pertenece cada una. Para que el transformador del producto sepa que tiene que interpretar las condiciones del fichero hay que marcar el *source element* que lo representa con el tipo *ps:pvsclxml* en el *Family Model*. Al realizar la transformación del producto, los atributos de las marcas se evaluarán dinámicamente y se decidirá si la marca (y las submarcas) entra o no en el fichero del producto final. El nombre del atributo que hay que

añadir a la marca que hay que evaluar porque pertenece a alguna *Feature* es *pv:condition* y se utiliza del siguiente modo:

```
<?xml version='1.0'?>
<text xmlns:pv="http://www.pure-systems.com/purevariants">
 <capitulo>
 Este es el capítulo uno.
 </capitulo>
 <autor pv:condition="Autor">
 Pepito.
 </autor>
</text>
```

Si la *Feature* "Autor" esta seleccionada el atributo de la condición será *true* y las marcas del autor aparecerán en el fichero del producto final.

```
<?xml version='1.0'?>
<text xmlns:pv="http://www.pure-systems.com/purevariants">
 <capitulo>
 Este es el capítulo uno.
 </capitulo>
 <autor>
 Pepito.
 </autor>
</text>
```

En cambio, si no se selecciona la *Feature* "Autor" la condición no se cumple y la marca desaparecería del fichero del producto final.

```
<?xml version='1.0'?>
<text xmlns:pv="http://www.pure-systems.com/purevariants">
 <capitulo>
 Este es el capítulo uno.
 </capitulo>
</text>
```

Etiquetación de ficheros con el mecanismo de sentencias condicionales

Para utilizar este mecanismo, el fichero tiene que tener el tipo *ps:pvscltext* asociado en el *source element* que lo representa. Para localizar las diferentes *Features* en el fichero se utilizan sentencias condicionales como *if*. Dentro del bloque de la sentencia se introduce el código que tiene que ser evaluado para saber si tiene que estar en el producto final.

Las sentencias que se utilizarán en el proyecto para anotar el código son las que aparecen en la tabla 4.1, *pure::variants* dispone de más sentencias, pero no se van a utilizar.

Sentencia	**Descripción**
PVSCL:IFCOND (condición)	Abre un bloque para una nueva condición. El código del bloque se incluye en el resultado final si la condición establecida se cumple. El bloque se debe cerrar con un PVSCL:ENDCOND.
PVSCL:ELSEIFCOND (condición)	Esta sentencia se utiliza después de utilizar una llamada PVSCL:IFCOND o PVSCL:ELSEIFCOND. Si la condición de la llamada que le precede no se cumple, se evaluará la condición del PVSCL:ELSEIFCOND. Si se evalúa y se cumple la condición el código que contiene estará en el producto final.
PVSCL:ELSECOND	La sentencia se usa detrás de una llamada PVSCL:IFCOND o PVSCL:ELSEIFCOND. Si la condición de la llamada anterior falla, se incluirá el código del bloque de esta sentencia.
PVSCL:ENDCOND	Cierra los bloques condicionales. Se puede utilizar después de una sentencia PVSCL:IFCOND, PVSCL:ELSEIFCOND o PVSCL:ENDCOND.

Tabla 4.1: Sentencias de *pure::variants* para asociar bloques de código con las *Features*.

Estas sentencias se pueden establecer en cualquier punto del fichero. Los bloques condicionales se pueden agrupar entre ellos, es decir, es posible que dentro de una condición se pueda meter otra condición. La segunda condición solo se evaluaría si la primera condición se cumple.

A continuación, se muestran algunos ejemplos del uso de estas condiciones:

```
1  public int calcularTemperatura(int t) { //temperatura como parametro en Celsius
2      // PVSCL:IFCOND(Kelvin)
3      // Celsius a Kelvin
4      t=t+273;
5      // PVSCL:ELSEIFCOND (Fahrenheit)
6      // Celsius a Fahrenheit
7      t=(t*1.8)+32;
8      // PVSCL:ENDCOND
9      return t;
10 }
```

En este ejemplo aparece un método cuyo objetivo es devolver el valor de la temperatura. Para ello, el usuario cuenta con tres *Features* diferentes para elegir la escala para medirla: “Kelvin”, “Fahrenheit” y “Celsius”.

Si el usuario ha elegido la *Feature* “Kelvin” la función que obtendrá será la siguiente:

```
public int calcularTemperatura(int t) { //temperatura como parametro en Celsius
    //
    // Celsius a Kelvin
    t=t+273;
    //
    //
    return t;
}
```

Si la opción escogida es “Fahrenheit” la función será la siguiente:

```
public int calcularTemperatura(int t) { //temperatura como parametro en Celsius
    //
    //
    // Celsius a Fahrenheit
    t=(t*1.8)+32;
    //
    return t;
}
```

Finalmente, si la opción es “Celsius” no se incluye código de ninguno de los bloques de sentencias:

```
public int calcularTemperatura(int t) { //temperatura como parametro en Celsius
    //
    //
    //
    return t;
}
```

También existe la posibilidad de añadir el *flag* LINE a la sentencia para que al evaluar la sentencia elimine toda la línea de código donde se encuentra la sentencia. Esta opción es muy conveniente para eliminar rastros de código. Se utiliza de la siguiente manera:

```
public int calcularTemperatura(int t) { //temperatura como parametro en Celsius
    // PVSCL:IFCOND(Kelvin, LINE)
    // Celsius a Kelvin
    t=t+273;
    // PVSCL:ELSEIFCOND (Fahrenheit, LINE)
    // Celsius a Fahrenheit
    t=(t*1.8)+32;
    // PVSCL:ENDCOND
    return t;
}
```

En la función que se obtiene tras la transformación no quedarán los rastros y el código quedará limpio. En el siguiente ejemplo se muestra como queda la función si la *Feature* elegida es “Kelvin”.

```
public int calcularTemperatura(int t) { //temperatura como parametro en Celsius
    // Celsius a Kelvin
    t=t+273;
    return t;
}
```

4.2.4. Generación de productos. VDM

Para generar un producto de la LPS en *pure::variants* se hace uso de los ficheros VDM que representan diferentes variantes de los productos que se pueden crear siguiendo el *Feature Model*. Para crear un producto, se deben seleccionar las *Features* que se consideren necesarias y que cumplan todas las restricciones. Una vez hecha la selección, se procede a hacer la transformación del modelo VDM a un producto. Al realizar la transformación, *pure::variants* resuelve los *ifdefs* de los ficheros *pvscltext*, las condiciones de los ficheros *condtex* y realiza la copia de los ficheros que sean necesarios desde el *Family Model* a la carpeta de destino del nuevo producto.

La aplicación que se genere a partir del VDM de la figura 4.11 tendrá la opción de registrarse y recibir mensajes, pero no podrá enviar mensajes. Además, el sistema de gestión de base de datos que se ha escogido para almacenar los datos es MySQL.

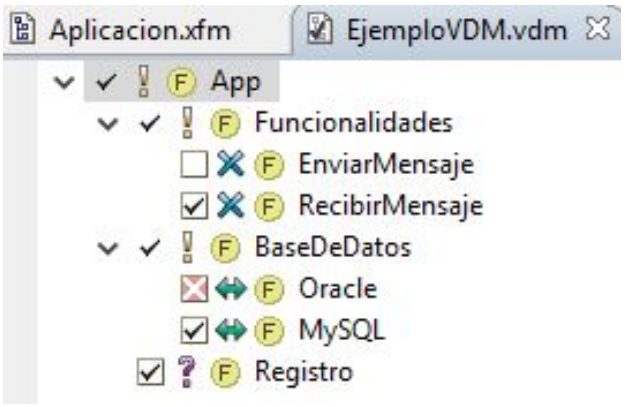


Figura 4.11: Ejemplo de la selección de *Features* en un fichero VDM.

Una vez adquiridos los conocimientos necesarios sobre las Líneas de Productos Software, entender que beneficios aporta a la producción de los productos con los que se cuenta hasta ahora y aprender a utilizar la herramienta *pure::variants*, es el momento de comenzar a realizar el diseño de la LPS.

5. CAPÍTULO

Productos de partida

En el presente capítulo se exponen los tres productos que se han analizado y tomado como referencia para realizar la LPS de anotaciones Web. Cada extensión ha sido desarrollada con una diferente intención, por lo tanto, las tres reúnen funcionalidades en común y algunas funcionalidades específicas para el objetivo para el que han sido producidas.

En los tres productos de partida se solapa la funcionalidad: algunas funcionalidades las tienen los tres productos, otras sólo las tienen algunos y otras solo se encuentran en un único producto (véase la figura 2.1). El objetivo del análisis de los productos es identificar estas funcionalidades para luego poder extraer los elementos comunes y variables que tienen entre los tres.

Las tres son extensiones de *Google Chrome* y están disponibles tanto en la *Chrome Web Store* como en GitHub. En el apartado final del capítulo se detallada la arquitectura que siguen las aplicaciones que es común en las tres.

5.1. Highlight&Go

Esta herramienta ha sido desarrollada con la intención de facilitar la actividad de extracción de datos en revisiones sistemáticas de la Literatura (RSL). Para realizar una revisión de este tipo, se debe hacer una extracción de datos sobre estudios primarios clasificándolos en base a unos criterios establecidos para el estudio. Gracias a la extensión *Highlight&Go* los datos se pueden almacenar, validar y visualizar. [Díaz et al., 2019b]

Las RSL se han utilizado en la ingeniería de software con el objetivo de identificar estudios y buscar los posibles vacíos que se pueden encontrar en ellos. La propuesta de la herramienta es convertir las extracciones de datos de los estudios primarios en recursos web abiertos, de manera que se puedan consultar y reutilizar tanto el proceso como los resultados de las RSL. Para ello, lo que propone es convertir los datos en anotaciones y almacenarlas en un repositorio de anotaciones Web, en este caso *Hypothes.is*.

Para que el usuario realice las actividades relacionadas con la revisión sistemática dispone de dos modos: el *Classifying mode* y el *Checking mode*.

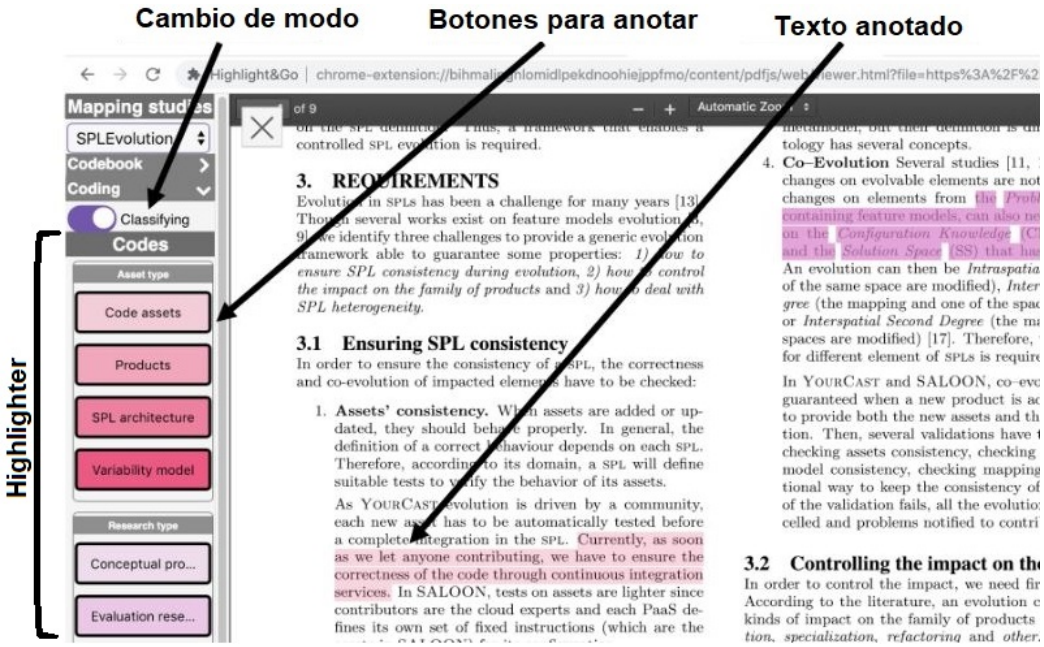


Figura 5.1: *Classifying mode* de *Highlight&Go*.

En el *Classifying mode* se realiza la extracción de datos mediante *coding*. Esta técnica consiste en examinar y organizar los datos que contiene cada estudio de la revisión sistemática. Para ello, hay que identificar líneas en el texto que ejemplifiquen la misma idea. La identificación de las ideas se realiza subrayando líneas del documento para posteriormente clasificarlas sobre un determinado criterio. El subrayado se realiza seleccionando las evidencias encontradas y seguidamente eligiendo mediante el botón correspondiente del *highlighter*[1] el criterio con el que se quiere clasificar esa parte del texto. Tras pulsar el botón que determinará sobre qué criterio está identificado el texto, las líneas que estaban

[1]Se trata de la barra que se encuentra a la izquierda de la capa de anotación que contiene todos los criterios o ideas en forma de botones de colores sobre los que se puede identificar un pasaje del documento.

seleccionadas se marcan con el color del botón que se ha seleccionado. A esta técnica se le denomina *colour-coding*.

En la figura 5.1 se pueden observar elementos mencionados en el párrafo anterior, que forman parte del *colour-coding*. A la izquierda de la figura se encuentra el *highlighter* que contiene los botones para anotar el texto. Los botones representan criterios como "Code assets", "Products" o "SPL architecture".

El texto que está anotado en el documento de la figura, esta clasificado con el criterio "Code assets". Esto se puede saber porque el color del subrayado es el mismo que el del botón del criterio.

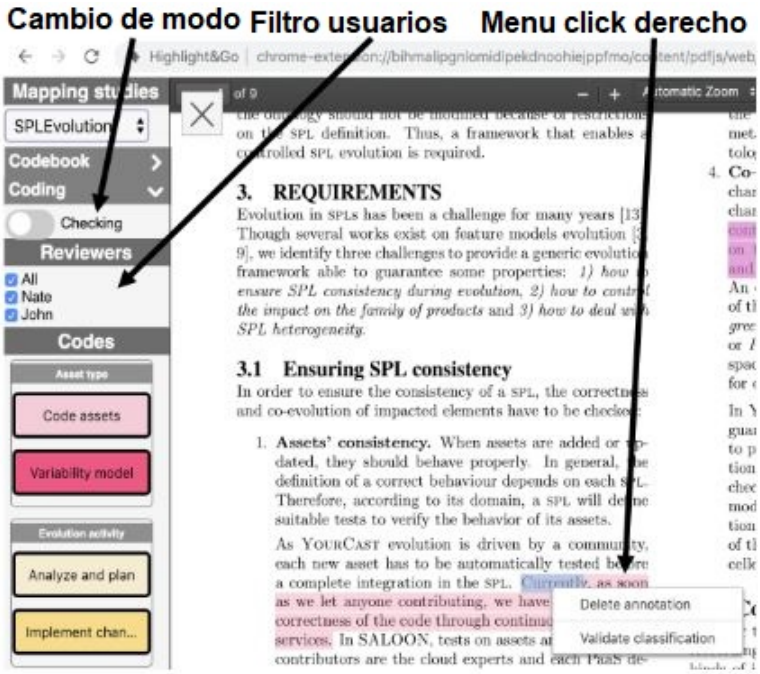


Figura 5.2: *Checking mode* de *Highlight&Go*.

Por otro lado, el *Checking mode* sirve para que el verificador pueda validar el contenido subrayado por los otros revisores. Para realizar la validación en este modo se dispone de un filtrado de las anotaciones en base a quién anotó el contenido que le permite visualizar las anotaciones de los revisores que le interesen. Para que el verificador del estudio primario valide un código, debe pulsar el botón derecho sobre el texto que esta subrayado y elegir la opción de validar.

La obtención de las ideas o criterios sobre los que se quiere evaluar el estudio primario se realiza en la hoja de cálculo que se quiere luego utilizar para mostrar los datos extraídos.

Para ello, en primer lugar, se debe configurar el *highlighter*, que es lo que sostendrá los botones para realizar el *colour-coding*, obteniendo así los diferentes criterios con los que extraer los datos del estudio.

Al convertir los datos en recursos web, permite que estos puedan ser consultados y tratados en el lugar donde se deben recoger los datos de manera automática. En el caso de esta extensión, durante la revisión a medida que los datos se van extrayendo y validando se plasman en la hoja de cálculo de *Google* correspondiente. La hoja de cálculo se convierte en un cliente del repositorio de *Hypothes.is* donde se guardan las anotaciones. La extensión va realizando consultas para comprobar que existen nuevas anotaciones y sincronizar los datos que se extraen en modo de anotación con las celdas de la hoja de cálculo sin intervención del usuario y mejorando así la fiabilidad al reducir los errores de la intervención manual.

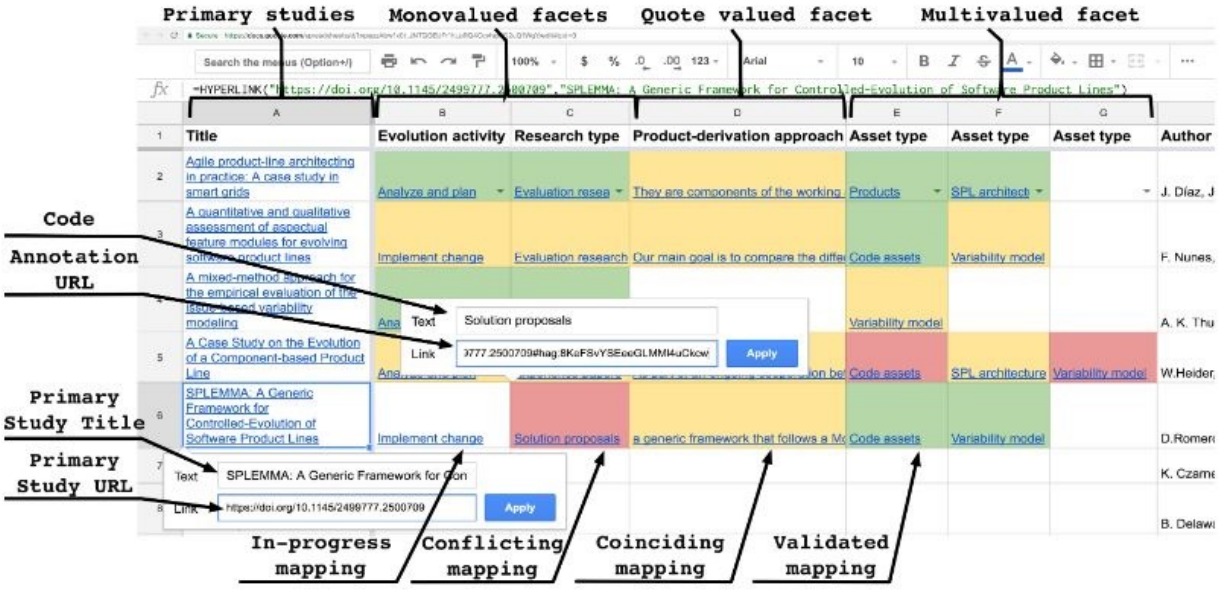


Figura 5.3: Vista generada en la hoja de cálculo a través de las anotaciones en *Highlight&Go*.

Funcionalidades de la extensión:

- Crear y borrar anotaciones.
- Crear los botones del *highlighter* mediante la hoja de cálculo.
- Seleccionar el grupo en el que se anota de manera manual.
- Filtrar las anotaciones en base a usuarios.
- Validar una anotación, es decir, que un revisor pueda verificar una anotación de otro revisor.
- Visualizar las anotaciones realizadas en la hoja de cálculo.

La versión de partida utilizada para *Highlight&Go* es la 0.1.15, pero puede que se utilicen funcionalidades de versiones nuevas que se desarrollen a la vez que el proyecto. La extensión se puede descargar a través de la *Chrome Web Store* o mediante la siguiente dirección de GitHub:

https://github.com/haritzmedina/highlightAndGo

5.2. Mark&Go

Esta herramienta ha sido implementada partiendo de una copia de *Highlight&Go* y sirve para facilitar las tareas de corrección y calificación de exámenes o ejercicios a los profesores, a través del uso de anotaciones Web en la plataforma de Moodle. *Mark&Go* es un *plugin* de *Chrome* que se integra con Moodle vía API y se utiliza en esta plataforma para la corrección de tareas en base a una rúbrica. La herramienta permite subrayar, comentar y evaluar el contenido de la tarea para que luego el estudiante pueda recibir su calificación y consultarla. [Mark&Go, 2019]

La manera común de los profesores para corregir los exámenes es descargar el archivo del estudiante desde el Sistema de Gestión de Aprendizaje, por ejemplo, Moodle o Google Classroom y hacer las correcciones localmente. Una vez corrigen la tarea deben subir la calificación y los comentarios de la tarea de manera manual de nuevo a la plataforma. La idea de *Mark&Go* es facilitar ese proceso al profesor, para ello la herramienta se encarga de recoger la rúbrica y crear unas anotaciones que luego permitan generar un *colour-coding highlighter* para que el profesor pueda ir encontrando evidencias en el ejercicio

con los criterios que ha definido en la rúbrica (véase la figura 5.4). El proceso de encontrar las evidencias se realiza subrayando la tarea basándose en el sistema de *colour-coding*. De esta manera, los elementos subrayados se convierten en anotaciones y se les permite darle una calificación.

En la figura 5.4 se observa como se mapea la rúbrica de Moodle en un *colour-coding highlighter* para *Mark&Go*. Para ello, la extensión al realizar la configuración de Moodle se encarga de obtener la definición de la rubrica y crear los elementos del *highlighter*. Las flechas de la figura muestran la relación entre la rúbrica y los elementos que se generan.

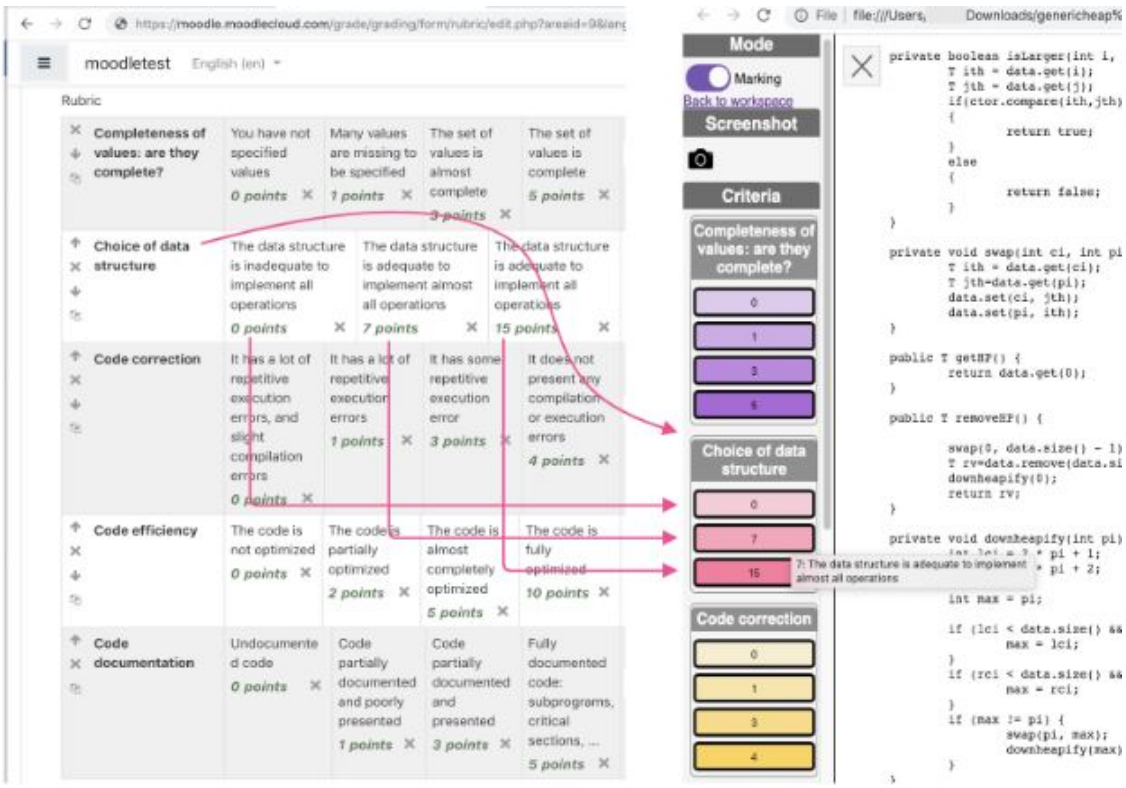


Figura 5.4: Generación del *highlighter* a partir de la rúbrica.

Gracias a *Mark&Go* las anotaciones realizadas en la tarea se van sincronizando con la página de la tarea de Moodle (véase la figura 5.5). La tarea plasma las calificaciones que se han dado al corregir la tarea en la rúbrica y a su vez los comentarios realizados sobre las evidencias encontradas se muestran también en la misma página con enlaces a la tarea para que el alumno pueda ver los comentarios en su contexto.

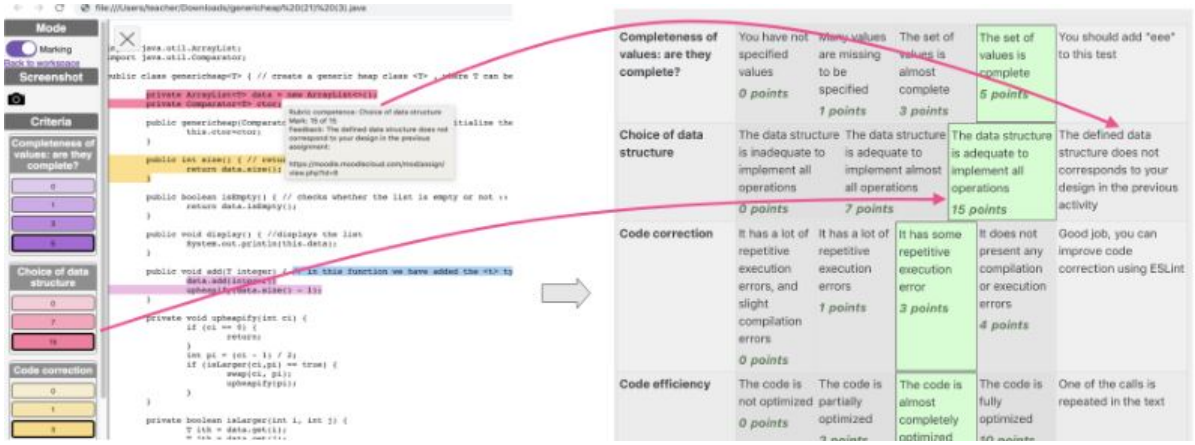


Figura 5.5: Resultados plasmados de la corrección en Moodle.

La idea de comentar las evidencias permite al profesor exponer la razón por la que ha dado esa calificación. Para realizar los comentarios la extensión proporciona facilidades como sugerir comentarios escritos anteriormente o añadir referencias de tareas anteriores.

Al igual que Highlight&Go 5.1, Mark&Go también dispone de dos modos: *Evidencing mode* y *Marking mode*. El *Evidencing mode* esta pensado para anotar las evidencias que se van encontrando en la tarea asociándolas a un criterio y el *Marking mode* permite darle la calificación que se ajusta a lo subrayado en el anterior modo.

Funcionalidades de la extensión:

- Crear y borrar anotaciones.
- Crear los botones del *highlighter* mediante la rúbrica de Moodle.
- Comentar una anotación.
- Responder a una anotación. El estudiante puede responder a un comentario realizado por el profesor.
- Visualizar los resultados de la corrección en Moodle.
- Sacar captura de pantalla al archivo que se esta corrigiendo incluyendo las anotaciones.
- Añadir referencias de tareas anteriores a los comentarios.
- Sugerir comentarios escritos anteriormente.

La versión de *Mark&Go* utilizada es la 0.1.9, pero puede que se utilicen funcionalidades de versiones nuevas que se implementen o cambien durante el proyecto. La extensión se puede descargar a través de la *Chrome Web Store* o mediante la siguiente dirección de GitHub:

https://github.com/haritzmedina/MarkAndGo

5.3. Review&Go

Review&Go se desarrolló a partir de una copia de *Mark&Go*. Hoy en día la revisión por pares tiene mucha demanda y los revisores no disponen de recursos suficientes para realizar el proceso de revisión. Con el objetivo de facilitar la tarea de revisar manuscritos se creó *Review&Go*, de esta manera, por medio de un *highlighter*, el revisor puede encontrar evidencias en el texto y marcarlas acorde a un criterio. [Díaz et al., 2019a]

La herramienta para realizar la revisión dispone de varias opciones que sirven para dar soporte a varios requisitos. Para realizar las anotaciones dispone de un *highlighter*, al igual que las otras herramientas. También se dispone de la opción de realizar una vista general de la revisión utilizando un canvas y un informe de las anotaciones realizadas. El *highlighter* es personalizable y el revisor lo puede amoldar a los criterios que desea utilizar. Además, también tiene la opción de colocarse en la última anotación para volver al último lugar donde ha trabajado.

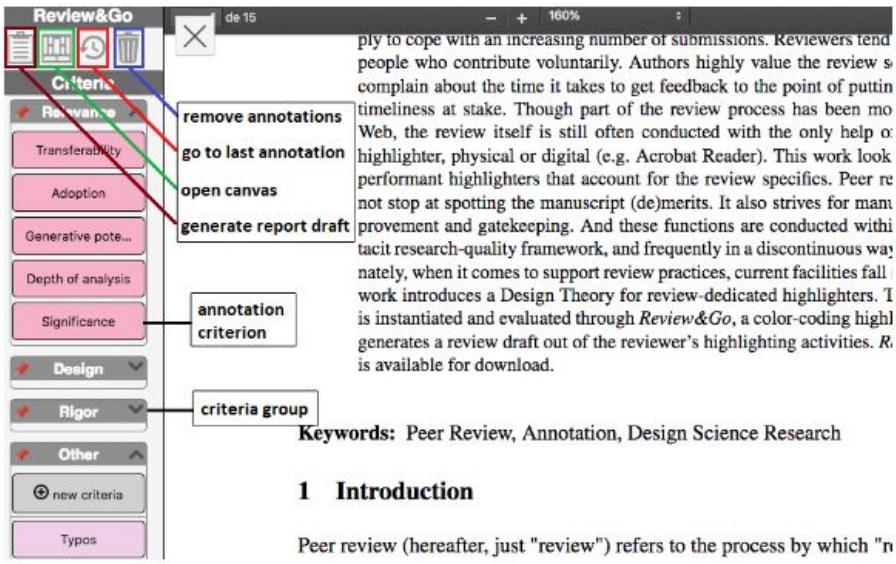


Figura 5.6: Interfaz de *Review&Go* a la hora de realizar la revisión.

Los criterios por defecto que forman el *highlighter* de la herramienta son los aspectos de revisión por pares con mayor acuerdo dentro de la comunidad DSR y estos se agrupan en tres grupos: Rigor, Revelance y Design. Estos grupos se encuentran en el *highlighter* y este les proporciona a cada uno un color diferente para poder anotar los criterios en el texto.

Si algún revisor desea añadir un criterio nuevo que no se encuentra entre los predefinidos tiene la opción de hacerlo manualmente añadiéndolo al grupo "Other".

Para realizar una retroalimentación de calidad, se pueden agregar comentarios al texto subrayado y se le puede añadir una categorización y una referencia de otro artículo.

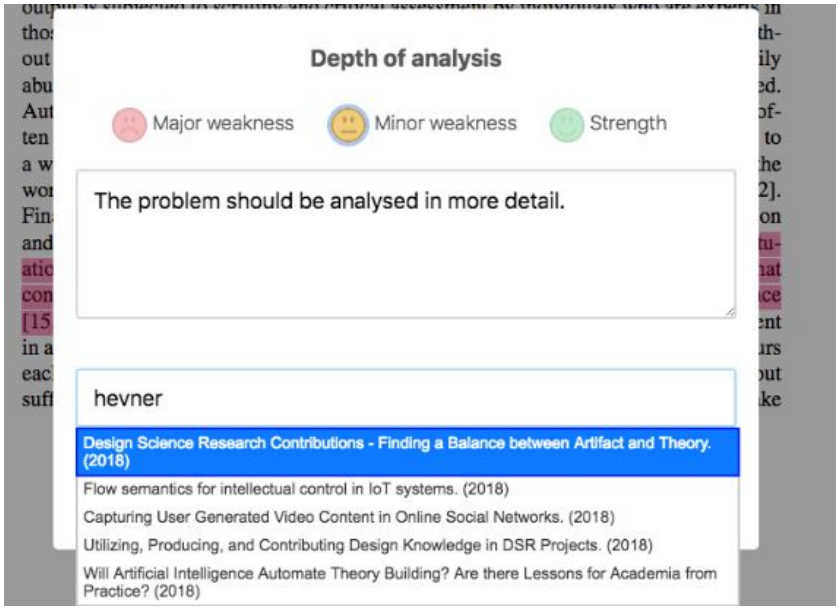


Figura 5.7: Comentar sobre una anotación en *Review&Go*.

Funcionalidades de la extensión:

- Crear y borrar anotaciones.
- Cargar los botones del *highlighter* mediante un archivo de configuración.
- Comentar una anotación.
- Realizar un canvas de las anotaciones.
- Realizar un resumen en texto de las anotaciones.
- Ir a la última anotación.

- Borrar las anotaciones realizadas.
- Añadir nuevos criterios.
- Sacar captura de pantalla al archivo que se está revisando incluyendo las anotaciones.
- Añadir referencias de artículos a los comentarios.
- Añadir una categorización al texto anotado.
- Consultar si un comentario puede resultar ofensivo a través de un análisis de sentimiento.

La versión de *Review&Go* utilizada es la 0.0.9, pero como en los productos anteriores, puede que se utilice alguna versión nueva que se desarrolle durante el proyecto. La extensión se puede descargar a través de la *Chrome Web Store* o mediante la siguiente dirección de GitHub:

https://github.com/haritzmedina/reviewAndGo

Figura 5.8: Ilustración de las funcionalidades encontradas en los productos de partida.

En la figura 5.8 se reunen las funcionalidades que se han encontrado tras probar las aplicaciones. Algunas funcionalidades las tienen los tres productos, otras sólo las tienen algunos y otras solo se encuentran en un único producto. La siguiente tarea será reunir estas funcionalidades en distintas *Features* para formar el *Feature Model*.

5.4. Arquitectura de las aplicaciones

Para poder realizar el análisis del código de los productos es importante saber qué estructura siguen y cómo están distribuidos sus archivos. Las aplicaciones nacen de una copia del producto anterior, por lo tanto, las tres mantienen la misma arquitectura y las diferencias entre ellas se encuentran dentro del código de algunos de los ficheros o en la existencia de ficheros nuevos que solo se encuentran específicamente para el producto.

Los tres productos son extensiones de *Chrome*, por lo tanto, son programas que se ejecutan en el navegador para añadir nuevas funcionalidades a la página. Las herramientas están formadas principalmente con tecnologías de desarrollo web: *HTML*, *JavaScript* y *CSS*, pero también contienen otros tipos de archivos como *JSON* o ficheros de imágenes. Los tres productos siguen la arquitectura común de las extensiones de *Chrome* que se presenta a continuación. [Google, 2019]

Las extensiones se crean con diferentes componentes. Toda extensión web debe contener un fichero **manifest** y entre los componentes de una extensión se pueden incluir **background scripts**, **content scripts**, la página de opciones y los **elementos UI**.

El fichero **manifest** se trata de un archivo JSON. Es el fichero más significante para el navegador ya que contiene la información más importante acerca de la extensión y se encarga de dirigirla. La información que contiene es el nombre de la extensión, la versión, los lugares donde se deben ejecutar los *scripts* y los permisos necesarios entre otras más cosas.

Los **background scripts** están pensados para supervisar los eventos y reaccionar a ellos con instrucciones especificadas. Los eventos son acciones que se completan en el navegador, como por ejemplo abrir o cerrar una pestaña cuando uno de los eventos ocurre, para ello, se debe ejecutar el *script* correspondiente.

Los **content scripts** son ficheros que se ejecutan en la página y son capaces de cambiar, añadir o eliminar elementos a través del uso del DOM[2].

[2]Document Object Model.

Por último los **elementos UI** son los encargados de personalizar la página mediante la interfaz de usuario. Con estos elementos se forma por ejemplo la barra izquierda que contiene el *highlighter* de las extensiones con las que se trabaja.

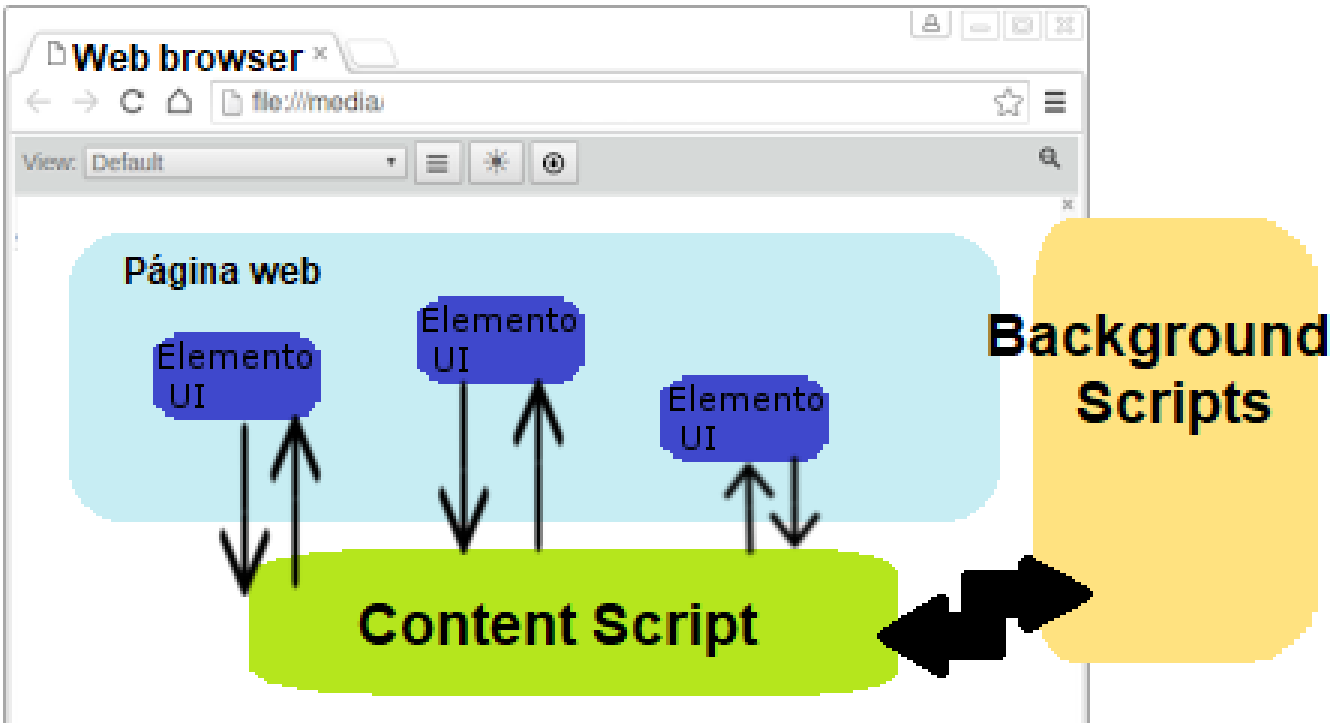


Figura 5.9: Arquitectura de una extensión web.

En la figura 5.9 se muestran los elementos mencionados anteriormente y como se relacionan dentro de la extensión. Los **background scripts** y los **content scripts** se comunican mediante mensajes. Los **content scripts** controlan los cambios que se producen en la página y son capaces de cambiar y gestionar los **elementos UI** que aparecen en la interfaz de la página web.

En la siguiente imagen se muestra el contenido de la carpeta de la aplicación, que es común en todos los productos.

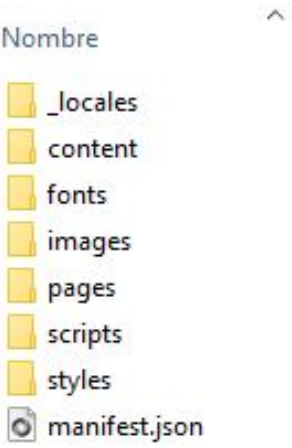


Figura 5.10: Contenido de la carpeta de la aplicación

Carpeta	**Contenido**
_locales	Contiene las diferente traducciones que existen para los idiomas de la aplicación. Actualmente solo se dispone del inglés.
content	Contienes los ficheros necesarios para cargar un visor PDF que utiliza la aplicación
fonts	No se utiliza en los productos. Esta carpeta se utiliza en el caso de que se quieran utilizar *fonts* (tipos de letra) en la extensión que no están por defecto en el navegador
images	Contiene las imágenes que se usarán en la aplicación
pages	Contiene los ficheros *HTML* que se utilizan en la aplicación.
scripts	Se encuentran todos los *scripts* que se ejecutan en la aplicación.
styles	Se encuentran las hojas de estilos que utiliza la aplicación.
manifest	Fichero que contiene la información detallada de la aplicación.

Tabla 5.1: Tipos de archivos que guarda cada carpeta de la aplicación.

Para buscar la variabilidad de los tres productos se debe encontrar cuales son los ficheros que cambian de un producto a otro. Tras analizar el contenido de las carpetas, lo que se debe tener en cuenta para realizar la extracción de *Features* porque su contenido no es el mismo en cada producto son: las carpetas **images** 5.2, **pages** 5.3, **scripts** 5.4, **styles** 5.9 y el fichero **manifest**.

En las siguiente tablas se muestra el contenido que tiene cada carpeta en conjunto de los tres productos y en qué aplicación se puede encontrar cada fichero. En el caso de que en la tabla aparezca el símbolo ✓significa que el fichero se encuentra en la aplicación, si

el símbolo es ✓* significa que el fichero se encuentra pero no se utiliza y si el símbolo se trata de una × significa que el fichero no se encuentra en la aplicación. Si se trata de un fichero que contiene código, en caso de aparecer en la aplicación, entre paréntesis se muestra el número de lineas que contiene. Uno de los objetivos con los que se realiza el proyecto es evitar la existencia de ficheros que no se utilizan, es decir, que no haya ficheros con el símbolo ✓*. Para ello la LPS se encarga de introducir cada fichero en la aplicación que lo necesite.

Fichero de images	**Highligth&Go**	**Mark&Go**	**Review&Go**
add.png	×	×	✓
AllowAccessFileUrlExtension.png	×	✓	✓
AllowAccessFileUrlScreenshot.png	×	✓	✓
append.png	×	✓	×
arrowDown.svg	✓*	✓*	✓*
arrowLeft.svg	✓*	✓*	✓*
arrowRight.svg	✓*	✓*	✓*
arrowUp.svg	✓*	✓*	✓*
deleteAnnotations.png	×	×	✓
generator.png	×	×	✓
icon-19.png	✓	✓	×
icon-19-bw.png	✓	✓	×
icon-38.png	✓	✓	✓
icon-38-bw.png	✓	✓	✓
icon-128.png	✓	✓	×
icon-128-bw.png	✓	✓	×
icon-256.png	✓	✓	✓
icon-256-bw.png	✓	✓	✓
icon-512.png	✓	✓	✓
icon-512-bw.png	✓	×	×
majorConcern.png	×	×	✓
minorConcern.png	×	×	✓
moodle.svg	×	✓	×
overview.png	×	×	✓
pin.png	×	×	✓
resume.png	×	×	✓
screenshot.png	×	✓	×
strength.png	×	×	✓
validate.png	✓	×	×
warning.png	✓	✓	✓

Tabla 5.2: Ficheros de la carpeta images.

En el caso de la imágenes, a la hora de construir la LPS se les añadirá una restricción con la *Feature* que esta enlazada, por lo que la imagen solo estará dentro del producto creado si la *Feature* ha sido elegida para su creación.

Fichero de pages (.html)	**Highligth&Go**	**Mark&Go**	**Review&Go**
sidebar	✓ (12)	✓ (12)	✓ (12)
annotatorMode	✓ (15)	✓ (15)	×
groupSelection	✓ (20)	✓ (11)	✓ (11)
tagWrapper	✓ (34)	✓ (35)	✓ (33)
userFilterWrapper	✓ (21)	✓* (21)	✓* (12)
popup	✓ (12)	✓ (12)	✓ (12)
options	✓ (18)	✓ (87)	✓ (83)
filePermission	×	✓ (40)	✓ (34)
toolset	×	✓ (11)	✓ (12)
reviewCanvas	×	×	✓ (39)
generator	×	×	✓ (12)

Tabla 5.3: Ficheros de la carpeta pages.

La mayor parte de los archivos *HTML* se encuentran en las tres extensiones. Algunos como *sidebar.html* es igual en los tres, otros como *groupSelection.html* solo les cambia un poco el contenido de dentro y algunos ficheros como *filePermission.html* solo aparecen en uno o algunos de los proyectos.

generator.html y toolset.html son dos ficheros diferentes pero tienen la misma función dentro de las herramientas, por lo tanto, sera necesario refactorizarlos en un único fichero.

Tal y como muestran los números de líneas de código que se encuentran entre paréntesis, se puede observar que todos los ficheros contienen un número de líneas parecido.

La carpeta con más contenido es la de scripts, ya que en los productos contiene un gran número de archivos y esta dividida en subcarpetas. Para mostrar los ficheros de la carpeta, en primer lugar, mediante una tabla, se mostrarán los archivos .js de la carpeta y seguidamente, por medio de más tablas, se mostrarán los ficheros de las carpetas de dentro de la carpeta scripts.

Fichero de scripts (.js)	Highligth&Go	Mark&Go	Review&Go
background	✓ (96)	✓ (114)	✓ (106)
Config	✓ (55)	✓ (57)	✓ (16)
contentscript	✓ (47)	✓ (55)	✓ (47)
options	✓ (5)	✓ (10)	✓ (10)
popup	✓ (11)	✓ (10)	✓ (10)
acmContentScript	✓ (85)	×	×
googleSheetsContentScript	✓ (27)	×	×
hypothesisGroupContentScript	✓ (46)	×	×
scienceDirectContentScript	✓ (31)	×	×
moodleAssignmentAddContentScript	×	✓ (19)	×
moodleContentScript	×	✓ (31)	×

Tabla 5.4: Ficheros .js de la carpeta scripts.

Fichero de scripts/background (.js)	Highligth&Go	Mark&Go	Review&Go
HypothesisManager	✓ (149)	✓ (149)	✓ (184)
DoiManager	✓ (94)	×	×
GoogleSheetsManager	✓ (72)	×	×
MoodleBackgroundManager	×	✓ (177)	×
MoodleDownloadManager	×	✓ (153)	×
TaskManager	×	✓ (168)	×
TourManager	×	✓ (50)	×
VersionManager	×	✓ (112)	×
CreateHighlighterTask	×	✓ (275)	×
Task	×	✓ (8)	×

Tabla 5.5: Ficheros de la carpeta scripts/background.

Fichero de scripts/hypothesis (.js)	Highligth&Go	Mark&Go	Review&Go
HypothesisClientManager	✓ (109)	✓ (124)	✓ (128)

Tabla 5.6: Fichero de la carpeta scripts/hypothesis.

Fichero de scripts/model (.js)	Highligth&Go	Mark&Go	Review&Go
Code	✓ (9)	×	✓ (9)
Facet	✓ (12)	×	×
MappingStudy	✓ (12)	×	×
AnnotationGuide	×	✓ (26)	✓ (26)
Criteria	×	✓ (77)	✓ (83)
GuideElement	×	✓ (40)	✓ (40)
Level	×	✓ (64)	✓ (64)
Rubric	×	✓ (133)	✓* (116)
Review	×	×	✓ (36)

Tabla 5.7: Ficheros de la carpeta scripts/model.

A continuación se presenta la carpeta de **content scripts** que es la que más ficheros contiene. La mayoría de ficheros son los mismos en los tres pero su contenido cambia mucho.

Fichero de scripts/contentScript (.js)	Highligth&Go	Mark&Go	Review&Go
AnnotationBasedInitializer	✓ (49)	✓ (69)	✓ (45)
ConfigDecisionHelper	✓ (34)	×	✓* (34)
ContentScriptManager	✓ (213)	✓ (231)	✓ (118)
ContentTypeManager	✓ (157)	✓ (230)	✓ (211)
Events	✓ (14)	✓ (17)	✓ (19)
GroupSelector	✓ (209)	✓ (123)	✓ (138)
ModeManager	✓ (106)	✓ (117)	✓ (116)
RolesManager	×	✓ (73)	✓* (26)
Sidebar	✓ (86)	✓ (86)	✓ (86)
TagManager	✓ (479)	✓ (484)	✓ (453)
UserFilter	✓ (254)	✓* (254)	✓* (254)
ContentAnnotator	✓ (12)	✓ (12)	✓ (14)
TextAnnotator	✓ (726)	✓ (989)	✓ (921)
RubricManager	×	✓ (40)	✓* (40)
Tag	×	✓ (39)	✓ (40)
TagGroup	×	✓ (16)	✓ (16)
Toolset	×	✓ (41)	×

Tabla 5.8: Ficheros de la carpeta scripts/contentScript.

Tras analizar el contenido parece que gran parte de las *Features* del Modelo de Características se podrán extraer de estos últimos scripts, y además, se predice que será necesario realizar una refactorización en ficheros como TagManager.js, que es diferente en los tres casos, para que las aplicaciones puedan ser compatibles y tener la misma base.

En las tablas observadas hasta ahora se han mostrado las carpetas que comparten las tres aplicaciones y que contienen ficheros parecidos que habrá que analizar para saber en que se diferencian. Además de estas carpetas, existen otras dentro de la carpeta de scripts, pero los ficheros que contienen ya son comunes en las tres extensiones o totalmente diferentes para funcionalidades especificas de cada extensión.

- **scripts/popup** y **scripts/options**: Cada una contiene un fichero de configuración de la extensión común en las tres.

- **scripts/utils**: Los ficheros son iguales en las tres extensiones. Se encuentran ficheros que contienen funciones para utilizar en los demás archivos *JavaScript*

- **scripts/specific**: Se encuentran ficheros específicos para cada una de las extensiones que proporcionan diferente funcionalidades.

- **scripts/googleSheets** y **scripts/googleSheetsContentScript**: Contienen los ficheros necesarios para la configuración de la hoja de cálculo que utiliza *Highlight&Go*.

- **scripts/moodle**: Contienen los ficheros necesarios para la configuración de Moodle que utiliza *Mark&Go*.

Por último, están los ficheros de hojas de estilo que se tratan de archivos del tipo .scss y se encargan de establecer el diseño visual y la interfaz de usuario.

Fichero de styles (.scss)	**Highligth&Go**	**Mark&Go**	**Review&Go**
contentScript	✓ (8)	✓ (12)	✓ (13)
groupSelection	✓ (20)	✓ (20)	× (20)
locationManager	✓ (60)	×	✓ (60)
options	×	✓ (12)	✓ (2)
pdfjs	✓ (21)	✓ (29)	✓ (25)
sidebar	✓ (122)	✓ (122)	✓ (122)
slrDataExtraction	✓ (12)	×	✓ (12)
tagWrapper	✓ (50)	✓ (64)	✓ (102)
textAnnotator	✓ (16)	✓ (11)	✓ (32)
variables	✓ (10)	✓ (10)	✓ (12)
exam	×	✓ (95)	×
modeManager	×	✓ (64)	×
toolset	×	✓ (14)	×
review	×	×	✓ (49)
reviewAssistant	×	×	✓ (48)
reviewCanvas	×	×	✓ (162)

Tabla 5.9: Ficheros de la carpeta styles.

Tras mirar el contenido de los tres proyectos, el siguiente paso es realizar la extracción de *Features* del código y realizar un Modelo de Características (*Feature model*) acorde al dominio de anotaciones Web. El resultado se encuentra en el siguiente capítulo.

6. CAPÍTULO

Análisis y diseño de la LPS:

Este capítulo plasma el análisis realizado sobre los tres productos base: *Highlight&Go*, *Mark&Go* y *Review&Go*. Al tratarse de una LPS que se desarrolla contando ya con varios productos, se utiliza un modelo de desarrollo extractivo, que en vez de comenzar la LPS desde cero, comienza a desarrollar la plataforma partiendo de los productos de la familia ya existentes. Para realizar esta tarea hay que empezar identificando las *Features* embebidas en el código de los productos.

Identificar las *Features* que se encuentran en el código fuente de los productos de partida no es una tarea sencilla. Al tener que analizar el código de tres productos se ha utilizado un mecanismo de comparación de código para identificar las líneas de código comunes entre los diferentes ficheros de los productos de partida. La aplicación que se ha utilizado para realizar la comparación del contenido de las carpetas y los ficheros es DiffMerge 2.6.4. Esta herramienta ha permitido la comparación uno a uno de los ficheros de los tres productos para analizar las líneas de código. El código común siempre es candidato a convertirse en un elemento común de la LPS. Por otro lado, las líneas de código específicas de cada producto son posibles *Features* que se encuentran dispersas por los productos.

En la figura 6.1 se muestra el resultado que se obtiene de la comparación entre una carpeta de *Mark&Go* y *Review&Go*. Se muestran los ficheros de las carpetas que no tienen el mismo contenido dentro, mientras que los ficheros que no se encuentran en las dos carpetas se ignoran.

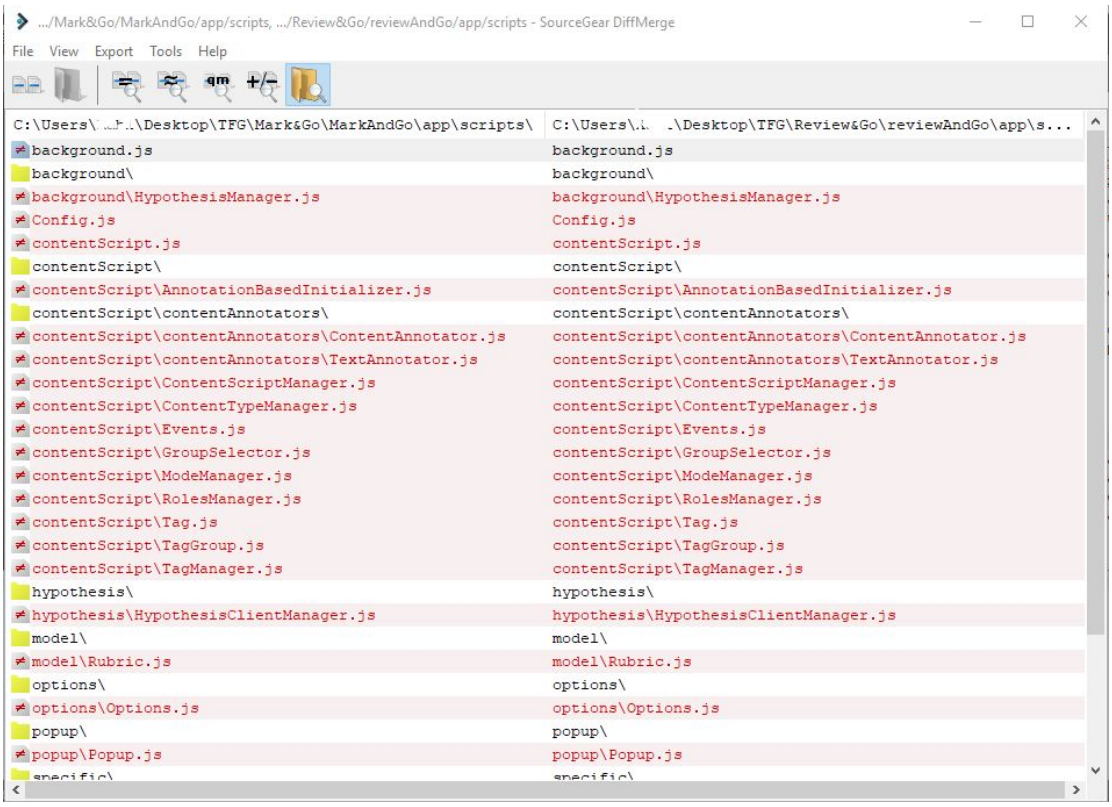


Figura 6.1: Comparación entre carpetas en DiffMerge.

Los ficheros de la izquierda pertenecen a *Mark&Go* y los de la derecha a *Review&Go*. Se realiza una comparación uno a uno de los ficheros con el mismo nombre de cada carpeta y si no contienen exactamente el mismo código, DiffMerge detecta un conflicto y se marcan en rojo.

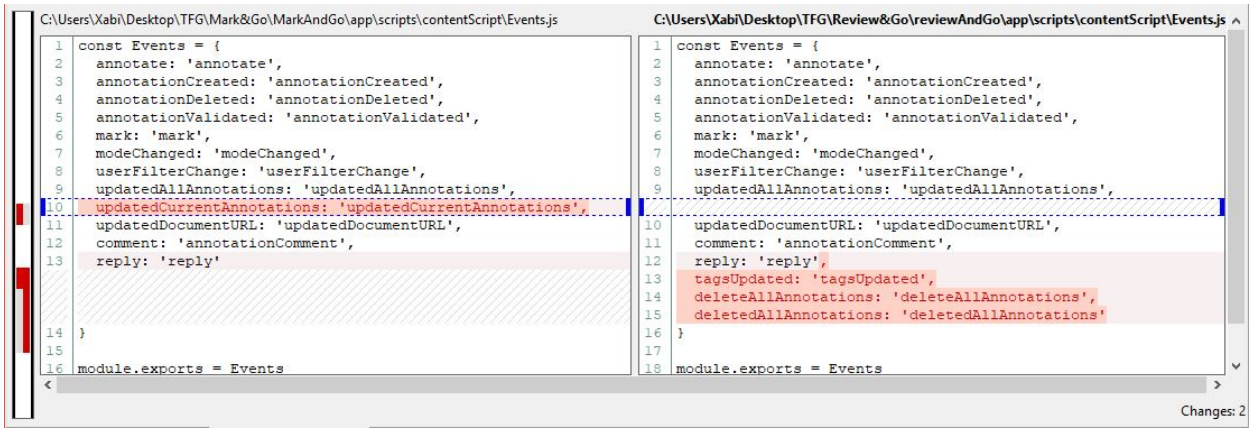


Figura 6.2: Comparación entre ficheros en DiffMerge.

Si se selecciona uno de los pares de ficheros, se abre una nueva ventana en la que se muestran los dos ficheros donde se resaltan las partes en las que no coinciden. En la figura 6.2 se muestra una captura de la ventana que se abre al seleccionar el par de ficheros Events.js de la carpeta contentScript que se veía en la figura 6.1.

En la comparación de ficheros anterior se puede observar que el fichero de la izquierda, el que pertenece a *Mark&Go*, contiene una línea que el de la versión de *Review&Go* no tiene y que esta última tiene tres líneas de código que el primer fichero mencionado no tiene. Estas líneas extras que tienen los dos ficheros son candidatas a ser parte de *Features* que son específicas para los dos productos.

Tras visualizar los ficheros con DiffMerge, se guarda el análisis y las *Features* identificadas. Para ello, se han realizado unas versiones PDF de los ficheros que ayuden a documentar el código analizado. Para plasmar el análisis en los documentos, en primer lugar, se ha realizado una copia de todos los ficheros de los tres productos en versión PDF. En segundo lugar, se ha abierto cada fichero PDF en el navegador y, por último, con la extensión de *Review&Go* se ha procedido a anotar las líneas de código. De esta manera, el fichero PDF contiene el código anotado con su respectiva posible *Feature*.

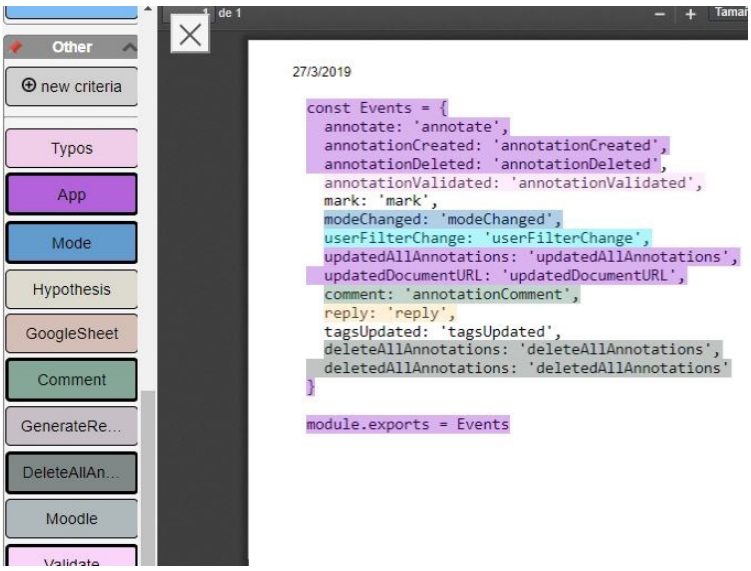


Figura 6.3: Extracción de *Features* de un fichero utilizando *Review&Go*.

En la figura 6.3 se muestra un ejemplo de cómo se ha anotado el fichero Events.js de *Review&Go* con dicha extensión. Mediante el botón App de color morado, se anota el código que es común en las tres aplicaciones y mediante los demás botones creados, se anotan las *Features* que se vayan identificando.

En la figura 6.4 se muestra un ejemplo de cómo se han organizado las carpetas de los productos de partida.

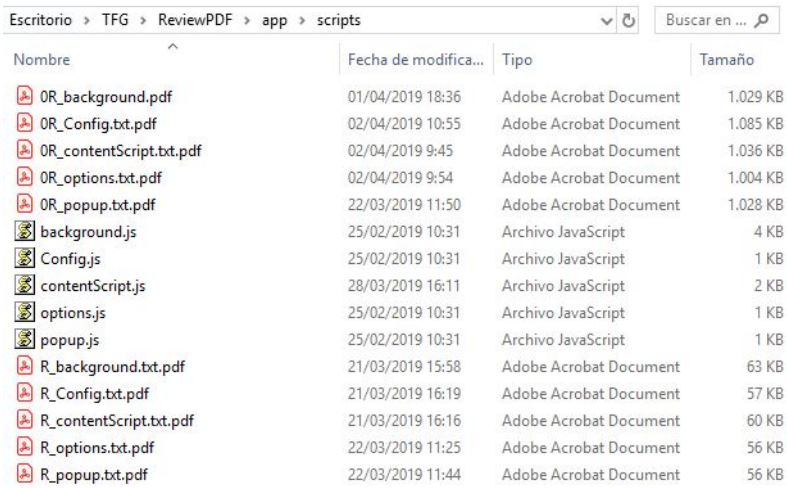
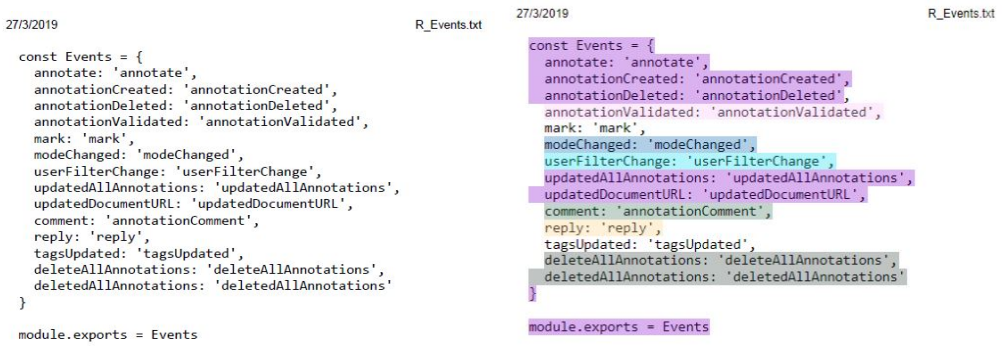


Figura 6.4: Ejemplo de la organización de una de las carpetas de uno de los productos.

En la carpeta de la captura anterior se aprecian tres versiones diferentes por cada fichero. En primer lugar, se encuentra la versión original, que en este caso se tratan de ficheros JavaScript. A partir de esta versión, se crea la versión PDF, que es la que se anota en el navegador. En la carpeta se identifica por el prefijo R (en el caso de *Highlight&Go* el prefijo es H y en *Mark&Go* es M). Una vez anotado el PDF anterior, se realiza una captura del documento y se consigue la tercera versión, que es el PDF con las anotaciones incluidas en el documento. Este PDF se identifica mediante el prefijo 0R en el caso de *Review&Go*, 0H en *Highlight&Go* y 0M en *Mark&Go*.



(a) R_Events.pdf **(b)** 0R_Events.pdf

Figura 6.5: Ejemplo del contenido de las dos versiones PDF de uno de los ficheros de los productos de partida

En la figura 6.5 se muestra un ejemplo de las dos versiones PDF que se guardan de un fichero correspondiente a *Review&Go*. El PDF de la izquierda 6.5a se crea para realizar anotaciones sobre él y conseguir la versión PDF de la derecha 6.5b

Tras analizar y anotar el código de las tres herramientas, se procedió a realizar el análisis y la definición de las *Features* que tienen los actuales productos. Además de las *Features* que se han encontrado en los productos, también se valoró la opción de incluir nuevas *Features* todavía no implementadas en los productos base, pero que son interesantes para futuras implementaciones. Como resultado del análisis se obtuvo el Modelo de Características (*Feature Model*) de la familia de productos.

Tras crear un primer prototipo del posible Modelo de Características, se realizaron varias reuniones entre el alumno, el tutor y el desarrollador de los productos para concretar un *Feature Model* definitivo. La siguiente sección detalla el *Feature Model* obtenido tras reunir las *Features* y agruparlas en las distintas propiedades que tiene una familia de anotadores Web.

6.1. Analisis del dominio

A la hora de diseñar un anotador Web hay que tener en cuenta los diferentes apartados que se encargan de gestionar la aplicación. Para distinguir los objetivos de las *Features* dentro de la Línea de Producto, se distribuyen en distintas secciones, que contienen un grupo de *Features* encargadas de cumplir con diferentes cometidos dentro de la aplicación. Estas secciones conforman el primer nivel del Modelo de Características de la LPS de anotadores Web 6.6.

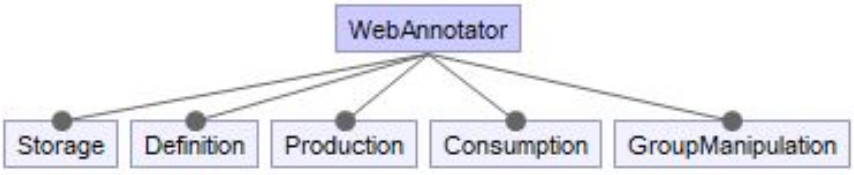


Figura 6.6: Primer nivel del Modelo de Características de anotadores Web.

Storage. Un anotador Web debe almacenar las anotaciones realizadas por los usuarios, por lo tanto, para poder guardar o recuperar las anotaciones, la aplicación debe contar con un sistema de almacenamiento. Las diferentes maneras de almacenaje de la aplicación se recogen dentro de esta *Feature*.

Definition. Cuando se crea la aplicación se debe definir el modelo de datos que se empleará en ella, para ello es necesario determinar cómo serán las anotaciones que se realizan, de qué recursos se pueden extraer y de dónde se obtienen los criterios sobre los que se realiza la anotación. Todas las *Features* relacionadas con lo mencionado anteriormente se agrupan dentro de esta *Feature*.

Production. Son varias las opciones que se pueden realizar o añadir sobre una anotación, bien a la hora de crearlas o una vez realizadas. En esta *Feature* se encuentra el grupo de *Features* que recogen las distintas formas de producir, gestionar y complementar las anotaciones.

Consumption. Crear anotaciones sobre un contenido puede servir para diferentes propósitos. Una vez producidas las anotaciones, los resultados de ellas se pueden plasmar e interpretar de distintos modos. Las *Features* relacionadas con consumir las anotaciones se encuentran dentro de esta *Feature*.

Group Manipulation. Se debe tener en cuenta que cada anotación se debe almacenar en un grupo junto a otras anotaciones, y juntas formarán todas las anotaciones que se realizan sobre un recurso. Se pueden aplicar varias opciones a los distintos grupos de anotaciones a la hora de gestionarlos, por lo tanto, en esta *Feature* se han agrupado estas distintas operaciones de manipulación del grupo.

En las próximas cinco secciones, se detalla cada apartado del primer nivel del *Feature Model* explicando sus funciones dentro del modelo y presentando el grupo de *Features* que contiene cada uno. Las *Features* que cuelgan de los elementos del primer nivel completan el *Feature Model* entero y a través de ellas, se podrá elegir qué *Features* tendrán los productos de anotadores Web generados por la LPS.

6.2. Storage

En una aplicación que realiza anotaciones, es fundamental que el usuario pueda disponer de alguna opción para almacenar las anotaciones que va realizando sobre los recursos

web. Poder almacenar las anotaciones en un lugar concreto permite que estas puedan ser recuperadas y utilizadas para diferentes fines en cualquier momento.

Existen dos principales alternativas para almacenar las anotaciones: es posible tener un almacén remoto (**Remote**) o almacenarlas en la misma extensión de Chrome de manera local (**Local**). El almacenaje remoto permite guardar las anotaciones en un almacén externo a la aplicación. Para ello, existen varias plataformas que proporcionan el sistema de almacenaje de las anotaciones, pero de momento, los productos de partida solo tienen implementada la posibilidad de almacenar las anotaciones en **Hypothes.is**.

En *Hypothes.is*, las anotaciones se guardan en un grupo de anotaciones. Un usuario puede disponer de múltiples grupos y tiene la posibilidad de compartirlos con más usuarios. Si se dispone de este sistema para almacenar anotaciones, los usuarios pueden gestionar y realizar todo tipo de búsquedas sobre las anotaciones realizadas desde la web de *Hypothes.is*.

Hypothes.is se trata de un proyecto relacionado con el mundo de la anotación Web y su objetivo es reunir anotaciones realizadas en contenidos de la Web. En *Hypothes.is*, las anotaciones se pueden guardar con etiquetas, que luego permiten agruparlas entre ellas o relacionarlas con el contenido de lo que se anota. Las etiquetas de las anotaciones también facilitan la realización de búsquedas de anotaciones.

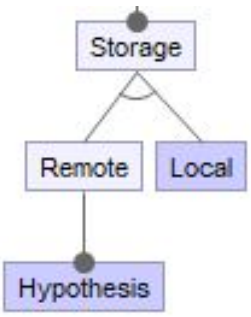


Figura 6.7: *Features* relacionadas con el Storage del Modelo de Características de anotadores Web.

En los productos de partida, solo existe la opción de *Hypothes.is* para guardar las anotaciones. El desarrollador de los productos ha estudiado la opción de implementar el almacenaje local en alguna de las extensiones porque permitiría realizar anotaciones sin necesidad de conexión a Internet y sin tener que compartir las anotaciones en la Web necesariamente. Además, en un futuro existe la posibilidad de incluir más alternativas para almacenar las anotaciones remotamente.

6.3. Definition

A la hora de definir el modelo de datos, se sigue la recomendación de W3C [1], que define la estructura de la anotación. Según el modelo de datos, [2] se deben tener dos conceptos en cuenta: un cuerpo, que contendrá el contenido que se anota (*Body*) y un identificador del recurso, a lo que se denomina *Target*. Teniendo en cuenta la composición de una anotación Web, en la definición se debe distinguir la información sobre el recurso anotado (**Target**) y el contenido de la anotación que se guarda mediante etiquetas (**Tag-based body**).

Dentro de la propiedad Target del Modelo de Características, se encuentran las distintas opciones que puede soportar la aplicación para identificar un recurso.

Target	**Descripción**
URL	Permite identificar los recursos Web mediante una dirección URL[3].
URN	Los URN[4] identifican recursos en la web, pero a diferencia de los URL, no indican exactamente dónde se encuentra ese objeto. Por lo tanto, si se cuenta con esta opción en la aplicación, se podrán identificar archivos que no se localizan en Internet, por ejemplo, los ficheros locales.
DOI	Permite elegir la opción de identificar los recursos mediante un identificador DOI[5] y poder realizar redirección a documentos mediante el identificador DOI.
NavigationScript	Da soporte a identificar artículos de varias páginas web en las que es posible realizar anotaciones. Actualmente se cuenta con la opción de identificar los artículos de los siguientes sitios web: **ACM**, **ScienceDirect** y **Dropbox**.

Tabla 6.1: Las *Features* que se agrupan dentro de Target.

Por otro lado, se debe definir cómo va ser el Tag-based body. Para definir la estructura del *Body*, la anotación se hará en base a una clasificación que permita reunir las anotaciones sobre la misma idea teórica o descriptiva. Las ideas se guardan en la anotación como

[1] [W3C, 2017a]

[2] Es recomendable haber leído el apartado 3.2 del capítulo sobre anotaciones Web para entender mejor cómo es el modelo de datos de una anotación Web.

[3] Localizador de Recursos Uniforme

[4] Nombre de Recurso Uniforme

[5] Digital Object Identifier

etiquetas y es posible clasificarlas sobre un nivel o dos niveles. Una anotación si se hace en base a un único nivel, solamente se guarda la etiqueta sobre cuál es el tema de la anotación (**Theme**), en cambio, existe la opción de incluir otras opciones distintas dentro de cada tema creando otro nivel (**Code**) para especificar más el tema de lo anotado. De esta manera, el usuario puede tener la opción de etiquetar la anotación solamente en base a un *theme* o a un *code*, que se agrupan dentro de los *themes*.

En las figuras 6.8 y 6.9 se pueden apreciar las dos formas que puede tomar el *highlighter* según su contenido. En la primera figura se observa un *highlighter* que dispone de distintos temas sobre los que se podrá etiquetar la anotación, y en el segundo caso, se encuentran los mismos temas que en la figura anterior, pero pudiendo establecer un nivel más (*code*) que permite especificar más el tema sobre el que se realiza la anotación.

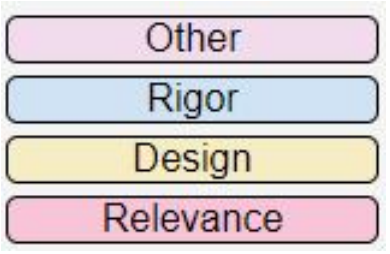


Figura 6.8: *Highlighter* que permite etiquetar en base a Themes.

Figura 6.9: *highlighter* que permite etiquetar en base a Themes y Codes.

En la figura de la izquierda, solamente se dispone de *Themes* con los que etiquetar la anotación: Revelance, Design, Rigor y Other. Por otro lado, en la figura de la derecha los *Themes* también disponen de *Codes*, por lo tanto, una anotación se puede etiquetar con uno de los *Themes* o uno de los *Codes* que se encuentra dentro de un tema. Puede ser que al contar con *Themes* y *Codes* alguno de los temas no tenga *Codes* dentro, como es el caso de Other.

Las etiquetas que se crean para poder clasificar las anotaciones pueden obtenerse tanto de manera estática como dinámica. La creación de etiquetas dinámicas (**Dynamic**), permite al usuario crear la etiqueta que desea en el tiempo de ejecución. Por otro lado, la creación de etiquetas estáticas (**Static**), crea las etiquetas al ejecutar por primera vez la aplicación.

En caso de querer utilizar etiquetas que están predefinidas, existen dos alternativas. La pri-

mera alternativa es crear las etiquetas en base a una configuración que realiza el usuario predefiniendo las etiquetas desde un fichero de configuración (**User**) que también puede ser importado en la aplicación. La segunda opción es recibir las etiquetas desde otra aplicación aparte mediante una API, que aporte la información necesaria para poder crearlas (**ThirdAppProvider**). Por ahora, se cuenta con dos alternativas que puedan proporcionar las etiquetas: las hojas de cálculo de Google (**GSheetProvider**) y el servicio de Moodle (**MoodleProvider**).

Mediante la opción de las hojas de cálculo de Google, al igual que en *Highlight&Go*, las etiquetas se obtienen de unas celdas especificas del documento. Por otro lado, en el caso de Moodle, tal y como se hace en *Mark&Go*, las etiquetas se crean mapeando la rúbrica obtenida de la tarea que se ha de configurar.

Al igual que en casos anteriores, ThirdAppProvider en un futuro se podrá dotar de muchas fuentes distintas con las que se podrán extraer las etiquetas requeridas para la aplicación.

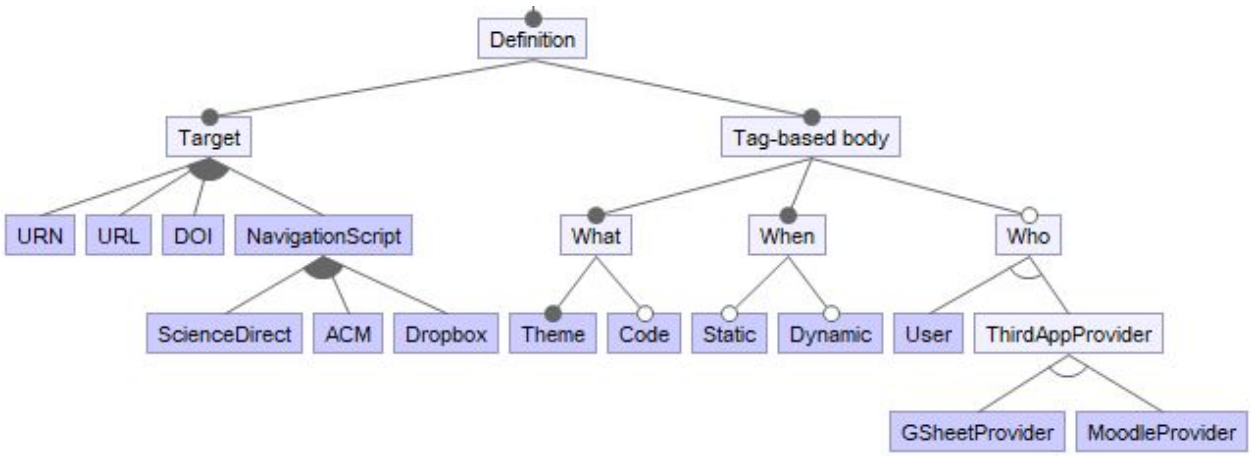


Figura 6.10: *Features* relacionadas con Definition del Modelo de Características de anotadores Web.

6.4. Production

Las operaciones de manipulación que se realizan sobre anotaciones se encuentran dentro del apartado de producción. Entre estas *Features* se encuentran las diferentes acciones que se pueden realizar sobre una anotación.

En la siguiente tabla se encuentran las *Features* que recogen dichas acciones.

Production	**Descripción**
Create	Recoge la funcionalidad de crear la anotación. Es una acción primordial para un anotador Web, por lo tanto, esta característica estará disponible en todos los productos de la LPS. Las anotaciones se pueden realizar de varias formas, pero de momento, solo esta implementada de realizar por medio de la técnica de *Colour Coding Highlight*.
Delete	Recoge la funcionalidad de borrar la anotación.
Validate	Permite la opción de validar una anotación realizada por otro usuario en el mismo recurso. Actualmente implementado en Highlight&Go pero es posible aplicarlo en más herramientas.
Reply	Da la opción de dar una respuesta a una anotación de un usuario y establecer una conversación con él.
Comment	Permite añadir un comentario a la anotación. Al hacer un comentario el usuario también tiene varias opciones para personalizar la nota que realiza sobre la anotación.

Tabla 6.2: Las *Features* que se agrupan dentro de Production.

En la siguiente figura se muestra un ejemplo de una aplicación que dispone de las opciones de borrar y comentar anotaciones.

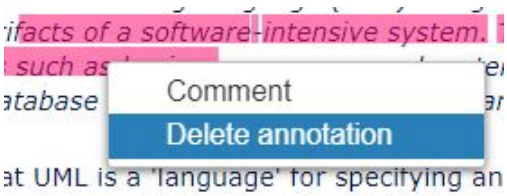


Figura 6.11: Ejemplo del menú de selección de la operación que se quiere realizar sobre la anotación.

Cuando un usuario realiza un comentario sobre una anotación tiene la opción de complementar el comentario realizado con distintas alternativas:

Production	**Descripción**
SentimentAnalysis	Al realizar un comentario sobre una anotación la aplicación detecta si se trata de un texto ofensivo, si es así lanza una advertencia.
AddReference	Permite añadir una referencia al comentario. El tipo de referencia varía según el contexto.
Categorize	Permite al usuario categorizar el comentario.
Autofill	Basándose en los comentarios que se han realizado anteriormente, a la hora de escribir un nuevo comentario, la aplicación le sugiere al usuario comentarios anteriores que sirven para autocompletar el texto. Esta funcionalidad es recomendable cuando se van a realizar comentarios con el mismo contenido.

Tabla 6.3: Las *Features* que se agrupan dentro de Comment.

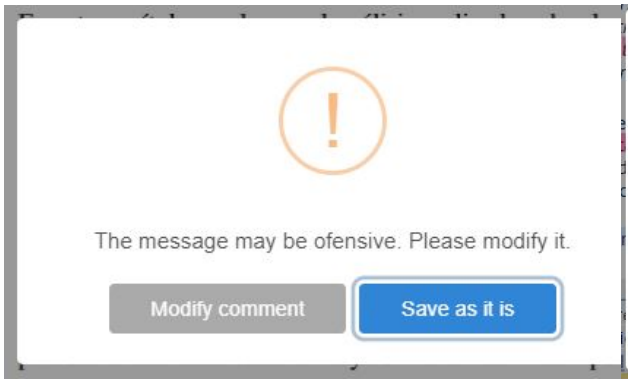


Figura 6.12: Advertencia que se lanza con SentimentAnalysis si el comentario es ofensivo.

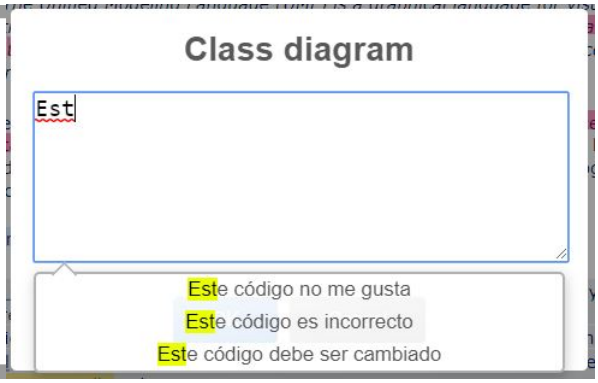


Figura 6.13: Interfaz de sugerencia de comentarios previamente utilizados, *Feature* Autofill.

Las *Features* relacionadas con la producción de anotaciones se estructuran dentro del *Feature Model* de la siguiente manera:

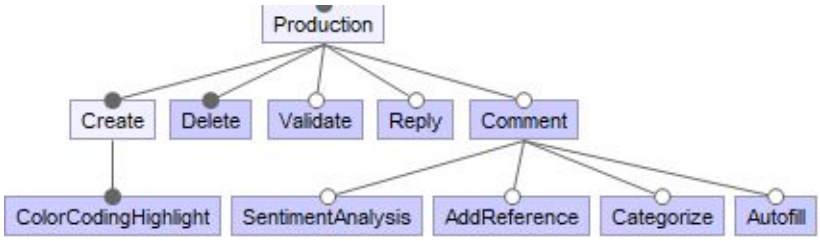


Figura 6.14: *Features* relacionadas con Production del Modelo de Características de anotadores Web.

6.5. Consumption

Consumption recoge las diferentes posibilidades con las que contará el usuario para poder ilustrar las anotaciones que se han ido realizando. El propósito de las anotaciones Web es recoger ideas del texto sobre un recurso para luego poder consumirlas y utilizarlas para varios fines.

Una vez se ha realizado el proceso de anotar el recurso, el usuario tiene la opción de consumir las anotaciones de distintas maneras y de obtener diferentes visualizaciones de ellas teniendo en cuenta sobre qué campo de aplicación se realizan.

En las aplicaciones generadas por la LPS, será posible realizar la consumición basándose únicamente en una única anotación (**SingleAnnotation**), teniendo en cuenta todas las anotaciones que forman el grupo donde se está anotando (**AnnotationGroup**) o recogiendo un conjunto específico de las anotaciones realizadas sobre un recurso, un conjunto de recursos o una consulta específica de anotaciones (**ResourceAnnotationCluster**). Los tres campos de aplicación que determinan las formas de recoger las anotaciones que se desean visualizar, se agrupan dentro de la *Feature* **Scope**.

Al igual que existen diferentes formas de agrupar las anotaciones que se quieren visualizar, también existen varias maneras de realizar la visualización de las anotaciones realizadas que se quieren ilustrar. Las diferentes opciones de visualizar las anotaciones se recogen en la tabla 6.4 y de momento, se cuenta con el conjunto de alternativas que formaban las tres aplicaciones de anotación Web con las que se ha realizado el proyecto.

Algunas de las visualizaciones que se realizan son muy específicas para los propósitos de dicha aplicación en la que se encontraba, por lo tanto, es posible que en pocos casos sean provechosas a la hora de crear nuevos productos.

Otro de los límites que se encuentra a la hora de visualizar el contenido es que los productos de partida, por ahora, aplican las visualizaciones solamente a un campo de anotaciones. Por ejemplo, a la hora de realizar una visualización canvas de las anotaciones tan solo se puede realizar sobre el campo ResourceAnnotationCluster, por lo tanto, se espera que en un futuro la Línea de Productos se adapte a aplicar las mismas visualizaciones sobre diferentes ámbitos.

Visualización	**Descripción**
TextSummary	Esta funcionalidad realiza un breve resumen sobre las anotaciones que se han realizado sobre una plantilla creada de manera predefinida. De momento solo aplicable con ResourceAnnotationCluster.
Canvas	Se realiza una ilustración de las anotaciones con una vista del modelo Canvas. De momento solo aplicable con ResourceAnnotationCluster.
Screenshot	Permite la opción sacar una captura del documento con las anotaciones realizadas y el resultado es un documento PDF con el recurso y las anotaciones realizadas.
ToolTip	Al situar el cursor sobre un elemento, en este caso una anotación, muestra información disponible sobre ella. De momento solo aplicable con SingleAnnotation.
LastAnnotation	Sitúa al usuario en el punto que ha realizado la última anotación. De momento solo aplicable con SingleAnnotation.
ThirdAppConsumer	Las visualizaciones se realizan en una plataforma independiente a la aplicación. Las aplicaciones ajenas consumen las anotaciones realizadas por los recursos mediante API y las ilustran en su contenido de la manera que se establezca. De momento existen dos aplicaciones que consumen las anotaciones que se realizan, se tratan de Google Sheet (**GSheetConsumer**) y Moodle (**MoodleConsumer**)

Tabla 6.4: Las *Features* que se agrupan dentro de Production.

Moodle consume las anotaciones para mostrar los comentarios que se han realizado y completar la rúbrica en la página de la tarea, al igual que en *Mark&Go*. Por otro lado, las hojas de cálculo de Google muestran las anotaciones que se han realizado en una nueva hoja en las celdas correspondientes.

Las visualizaciones Screenshot, Canvas y TextSummary se realizan por medio de unos botones que se encuentran en la caja de herramientas (*toolset*) de la barra lateral izquierda de la extensión.

Figura 6.15: De izquierda a derecha, los botones que realizan la función de Screenshot, Canvas y TextSummary.

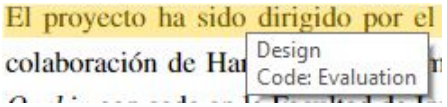


Figura 6.16: Ejemplo de lo que se visualiza mediante el tooltip al posicionarse sobre una anotación Web.

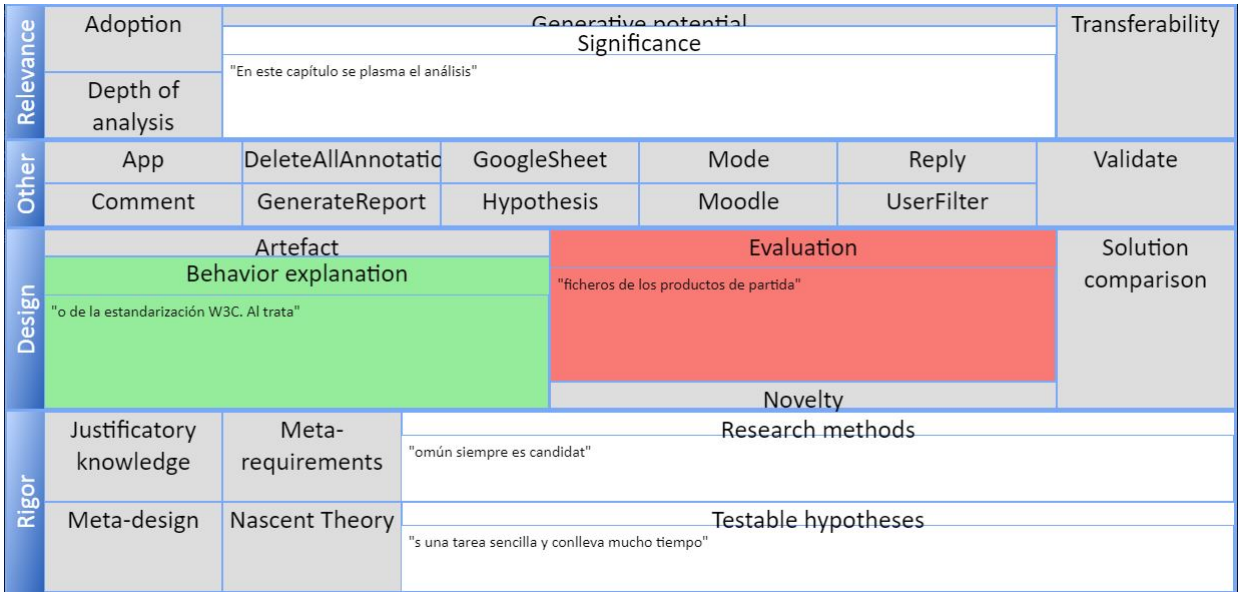


Figura 6.17: Visualización de las anotaciones a través de una vista Canvas.

Sobre las anotaciones realizadas también existe la opción de realizar un sistema de filtrado para buscar anotaciones en base a un criterio. Por ejemplo, se pueden realizar búsquedas de anotaciones aplicando un filtro usuario o grupo de usuarios (**ByUser**). De esta manera, se visualizan las anotaciones solamente realizadas por los usuarios que se han elegido en el filtro.

Hasta ahora el único tipo de filtrado que esta implementado es el mencionado anteriormente, pero es posible que en un futuro se implementen más basándose en diferentes

criterios, por ejemplo, sería posible realizar un filtro sobre la hora que se realizó la anotación (**ByTime**) o en base a la etiqueta que se ha utilizado al anotar (**ByTag**). El filtrado permitirá al usuario realizar diferentes búsquedas que le ayuden a encontrar las anotaciones que desea, se asemeja a hacer una selección en una base de datos.

Figura 6.18: Sistema de filtrado de anotaciones en base a usuarios.

El filtrado de anotaciones de la figura 6.18 provoca que solo se visualicen las anotaciones realizadas por Haritz y Xabi.

En la siguiente figura se puede observar cómo se recogen las *Features* dentro de Consumption.

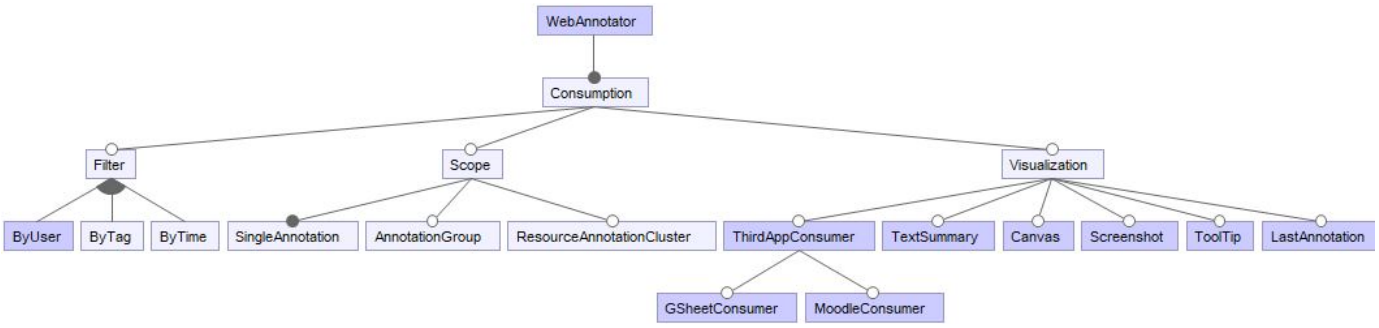


Figura 6.19: *Features* relacionadas con Consumption del Modelo de Características de anotadores Web.

6.6. GroupManipulation

Dentro de esta sección se encuentran las *Features* relacionadas con los grupos de anotaciones. Las anotaciones que se realizan sobre un recurso, se almacenan en grupos, y existen varias operaciones dentro de las aplicaciones que permiten gestionar estos grupos. La acción principal y fundamental que se debe realizar es crear el grupo sobre el que se

realizan las anotaciones (**CreateGroup**), por lo tanto, esta *Feature* es obligatoria en todas las aplicaciones.

La aplicación debe trabajar siempre sobre un grupo de anotaciones, pero un usuario puede disponer de diferentes grupos, por lo tanto, debe existir una manera de que el anotador del contenido seleccione un grupo en el que anotar (**SelectGroup**). Para la selección del grupo existen diferentes alternativas, la primera opción de seleccionar el grupo es que la misma aplicación escoja el grupo de anotaciones que se le haya predefinido (**ApplicationBased**), la segunda opción es que al usuario se le visualicen los grupos que tiene disponibles a través de una lista desplegable y pueda escoger manualmente sobre cual quiere trabajar (**Manual**) 6.20. Por último, existe la posibilidad de que la misma aplicación escoja el grupo automáticamente en base al recurso que se está utilizando para realizar las anotaciones (**ResourceBased**). Un ejemplo de ResourceBased es el sistema que se utiliza con Moodle en *Mark&Go*.

Figura 6.20: Lista desplegable de la *Feature* Manual.

Una aplicación generada a través de la LPS puede incluir la opción de borrar todas las anotaciones realizadas en el grupo de anotaciones (**DeleteGroup**). Esta opción permite al usuario borrar las anotaciones que recoge el recurso sobre el que está anotando.

Figura 6.21: Botón con el que se realiza la acción de borrar todas las anotaciones.

Al igual que en las demás *Features* de primer nivel, en **GroupManipulation** también existe la posibilidad de que en un futuro se puedan crear mas alternativas para completar

las aplicaciones. Una de las ideas que se baraja para próximas mejoras es que se puedan exportar e importar grupos (**ImportGroup** y **ExportGroup**). Además, una opción interesante también sería poder compartir el grupo en el que se está trabajando para que los usuarios puedan anotar sobre el mismo grupo (**SharingGroup**).

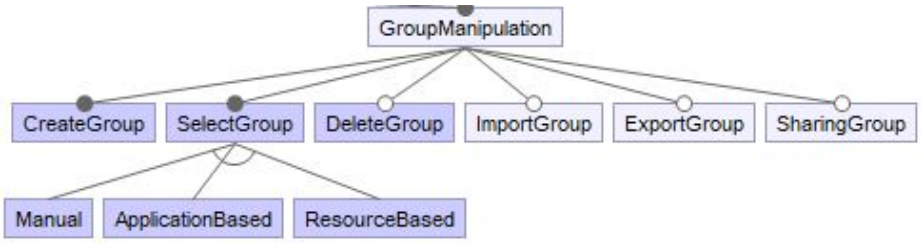


Figura 6.22: *Features* relacionadas con GroupAnnotation del Modelo de Características de anotadores Web.

Finalmente, con este último grupo de *Features* se cierra el *Feature Model*. En total se cuenta con 65 *Features*. 22 de las *Features* se utilizan para agrupar otras *Features* y tener el *Feature Model* mejor estructurado. Por otro lado, se cuenta con 5 *Features* (Hypothesis, URL, ColorCodingHighlight, DeleteAnnotation y CreateGroup) que son comunes en todos los productos y en un principio no van a tener implementación en la LPS, a no ser que en la fase de implementación se implementen más *Features* que aún están sin implementar en los productos.

6 de las *Features* que se han incluido al *Feature Model* son *Features* que aún no están implementadas en los productos pero puede que se implementen durante el desarrollo del proyecto y se puedan incluir en la LPS. Entre ellas se encuentran las *Features* Local, ByTag, ByTime, ImportGroup, ExportGroup y SharingGroup.

Las 32 restantes son *Features* que requieren implementación en el *pure::variants*. Esto significa que si a la hora de crear un producto se selecciona una de estas *Features*, se le va añadir una funcionalidad o una característica nueva al producto generado. Estas *Features* son las que se encuentran en los últimos nodos del *Feature Model*.

Una vez terminado el *Feature Model*, el siguiente paso es trasladarlo a la herramienta de *pure::variants* y empezar a implementar las *Features* para comenzar a generar productos.

7. CAPÍTULO

Implementación

Una vez realizadas las tareas relacionadas con la identificación de *Features* y disponer del *Feature Model* que compone la Línea de Productos de anotadores Web, se ha procedido a realizar la implementación de la LPS en *pure::variants*.

En el proceso de implementación, la tarea principal es la refactorización de *Features* que consiste en facilitar la variabilidad en la LPS. Para realizar la implementación se ha tenido que generalizar el código de los productos de partida para que sea compatible con las demás *Features* (Refactorización). En este proceso también es necesario configurar ficheros que formarán parte de la LPS, por ejemplo, el *Feature Model* para establecer la variabilidad de la LPS y el *Family Model* para que los productos se generen con los ficheros correspondientes. En la siguiente sección, se recogen los pasos que se han seguido para implementar la LPS en *pure::variants*. [1]

7.1. Preparación del entorno de trabajo

Para comenzar a realizar la implementación, se han seguido varios pasos para conseguir preparar el entorno de trabajo. Los pasos se han realizado uno detrás de otro tal y como se muestra en el siguiente diagrama de actividades 7.1.

[1] Para entender conceptos claves y pasos del desarrollo se recomienda haber leído antes el apartado 4.2

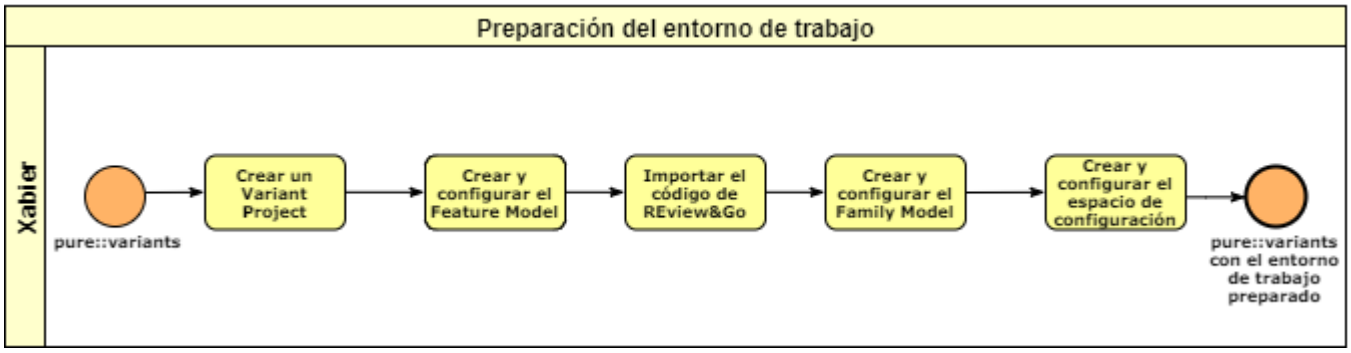


Figura 7.1: Actividades a realizar para preparar el entorno de trabajo.

La mayoría de pasos para preparar el entorno consisten en configurar ciertos componentes. En la siguiente figura 7.2, se muestran los elementos que se deben crear y configurar en la LPS.

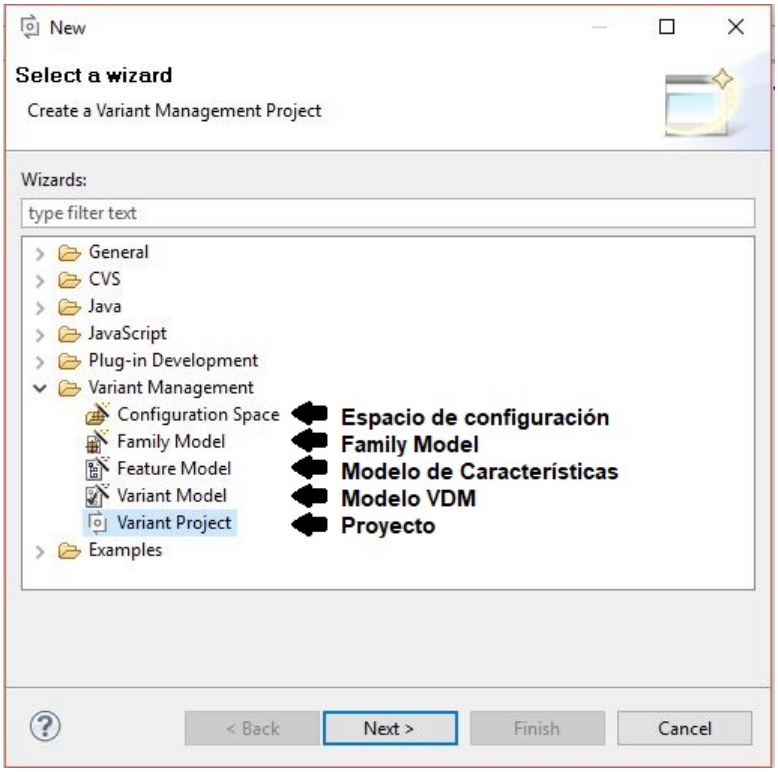


Figura 7.2: Ficheros que se deben configurar para realizar la LPS.

El primer paso ha sido preparar el entorno de trabajo en *pure::variants*. Para ello, en primer lugar, se ha creado un nuevo proyecto (*Variant Project*), donde se almacenan todos los ficheros de la LPS. El siguiente paso ha sido crear dos directorios: input y output.

En el directorio input se guardan todos los ficheros que pueden aparecer en los productos que se generan. A partir de estos ficheros, se forman las aplicaciones que se crean a través de la LPS. Por otro lado, en la carpeta output se almacenará el código de todos los productos que se generen a la hora de realizar las transformaciones.

El siguiente paso ha sido configurar el *Feature Model* del proyecto basándose en el diseño del capítulo anterior. Para ir creando el árbol que forma el *Feature Model*, se debe seleccionar la *Feature* padre de la nueva característica que se quiere añadir y elegir la opción de crear una nueva *Feature*. Al indicar que se quiere añadir una nueva *Feature* al modelo, se abre una ventana en la que hay que especificar sus propiedades: el nombre único, el nombre visible, el tipo de *Feature* (obligatoria, opcional, disyunción exclusiva o disyunción inclusiva), las restricciones, etcétera. Si a lo largo del desarrollo se requiere realizar cambios, es posible cambiar las propiedades de la *Feature* en cualquier momento escogiendo sobre ella abrir la ventana de propiedades.

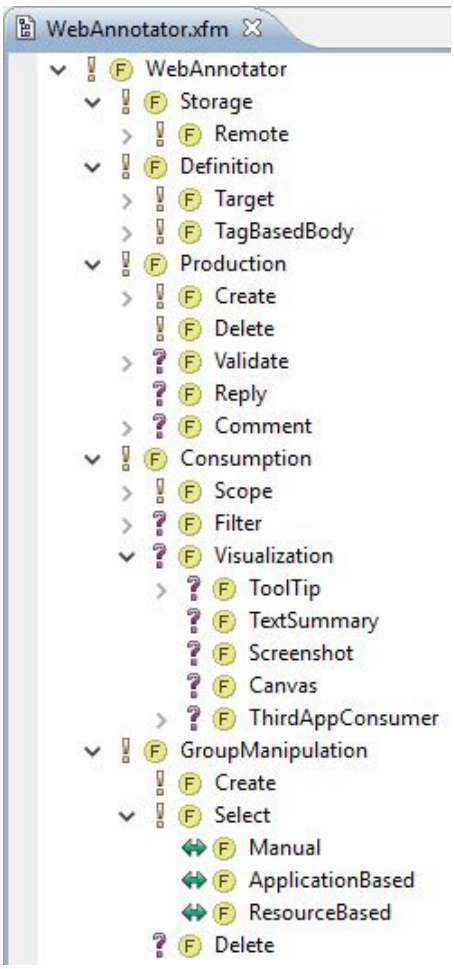


Figura 7.3: Parte del *Feature Model* del proyecto en *pure::variants*

En la figura 7.3 se muestra el *Feature Model* del proyecto en *pure::variants*, donde se puede observar la mayor parte de las *Features* que lo forman. En realidad, el *Feature Model* contiene todas las características que se han podido implementar del diseño mostrado en el apartado 6.1.

Después de realizar el *Feature Model* se ha procedido a realizar el *Family Model*, pero para ello es necesario tener ficheros con los que generar los productos. Para comenzar con la implementación de *Features* se han introducido los ficheros de *Review&Go* en la carpeta input del proyecto. Se ha decidido empezar a implementar las *Features* de *Review&Go* porque es el producto menos complejo de los tres y al tratarse del último producto de la familia, tiene las últimas versiones del código de los elementos comunes.

Una vez copiado el código, se ha realizado el *Family Model*. Elaborar un *Family Model* introduciendo los elementos uno a uno resulta una tarea muy costosa porque el proyecto cuenta con una gran cantidad de archivos. Para facilitar está labor, *pure::variants* cuenta con un *plug-in* llamado Connector for Source Code Management.[2] El *plug-in* se trata de un conector que permite tener sincronizada toda la estructura de los directorios y los ficheros de código fuente fácilmente.

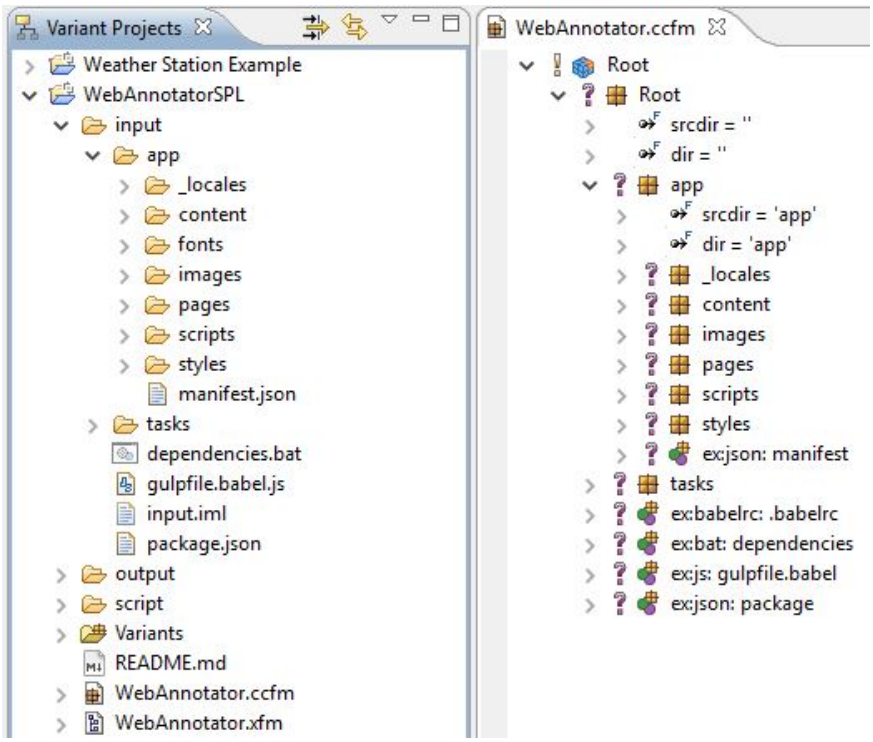


Figura 7.4: Estructura y *Family Model* del proyecto.

[2] [pure systems, 2019a]

A través de este *plug-in* se ha generado la primera versión del *Family Model*. A la hora de ir implementando nuevas *Features* y tener que introducir ficheros nuevos al código fuente del proyecto, no ha resultado necesario crear el *Family Model* de nuevo ya que el *plug-in* permite actualizar el estado del *Family Model* sincronizandolo con la carpeta donde se encuentran los ficheros del proyecto.

En la figura 7.4 se muestran dos ventanas. En la ventana de la izquierda se encuentran los proyectos de Líneas de Productos. La ventana permite navegar entre los ficheros que contienen y realizar acciones como mover o borrar los ficheros.

Por otro lado, en la ventana de la derecha se encuentra el fichero correspondiente al *Family Model* de la LPS. Tal y como se puede observar, el modelo tiene la misma estructura de la carpeta input del proyecto gracias al uso del plug-in de *pure::variants* que ayuda a mantenerlos sincronizados. Si extendemos las carpetas del proyecto y del *Family Model* se encuentran los ficheros del código fuente. En el *Family Model* se encuentra la representación de las carpetas y ficheros donde especifica cómo se debe tratar cada uno a la hora de realizar la transformación. Por ejemplo, en la figura 7.5 se muestra la representación de la carpeta de los ficheros HTML en el *Family Model*.

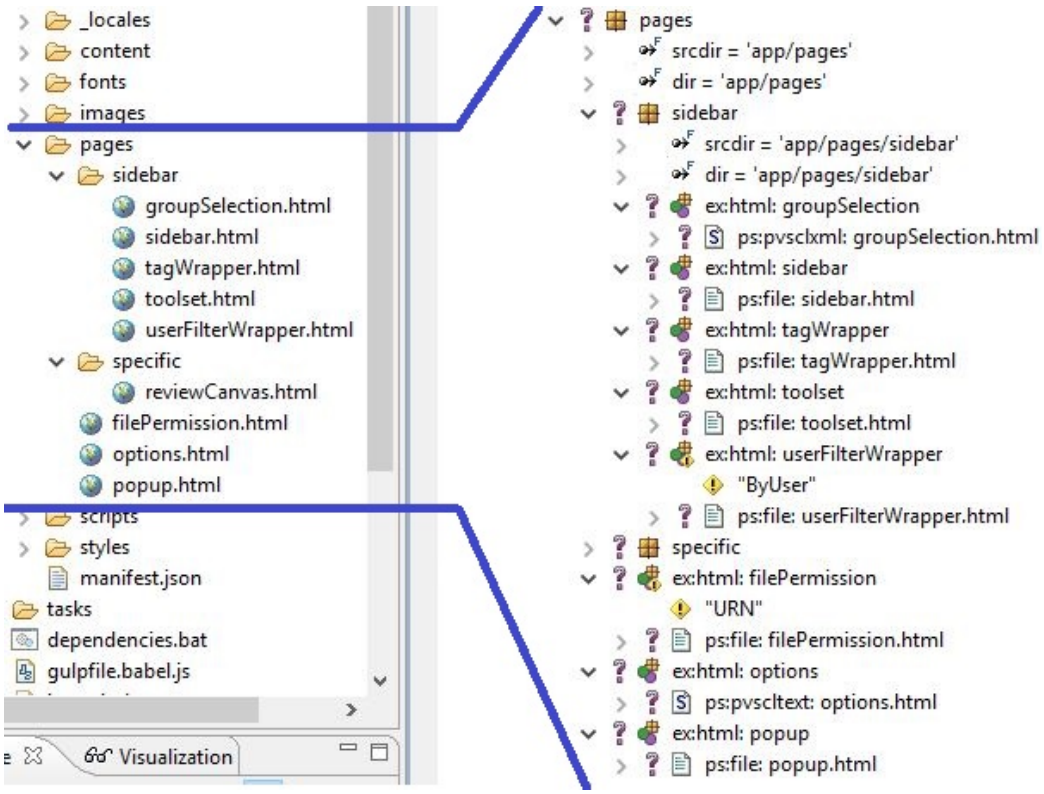


Figura 7.5: Carpeta del proyecto con el código fuente y su mapeo en el *Family Model*.

El proyecto de la LPS de anotadores Web es el denominado WebAnnotatorSPL y en la figura 7.4 se muestra cómo está estructurado. Como anteriormente se ha mencionado, se puede observar que contiene la carpeta input con el código fuente de los posibles productos y una carpeta output donde se almacena el código de los productos generados. En la carpeta script se encuentra un fichero necesario que proporciona *pure::variants* para realizar la transformación de JavaScript.

La carpeta Variants se trata de un espacio de configuración (*Configuration Space*). En este espacio se crean los modelos VDM para realizar las diferentes transformaciones de los productos.

Al crear este espacio, es necesario configurarlo, y para ello hay que indicar qué modelos se deben utilizar (el fichero del *Feature Model* y el del *Family Model*) para realizar las transformaciones de los VDM. En este caso los modelos que se utilizan son los ficheros WebAnnotator.ccfm (*Family Model*) y WebAnnotator.xfm (*Feature Model*).

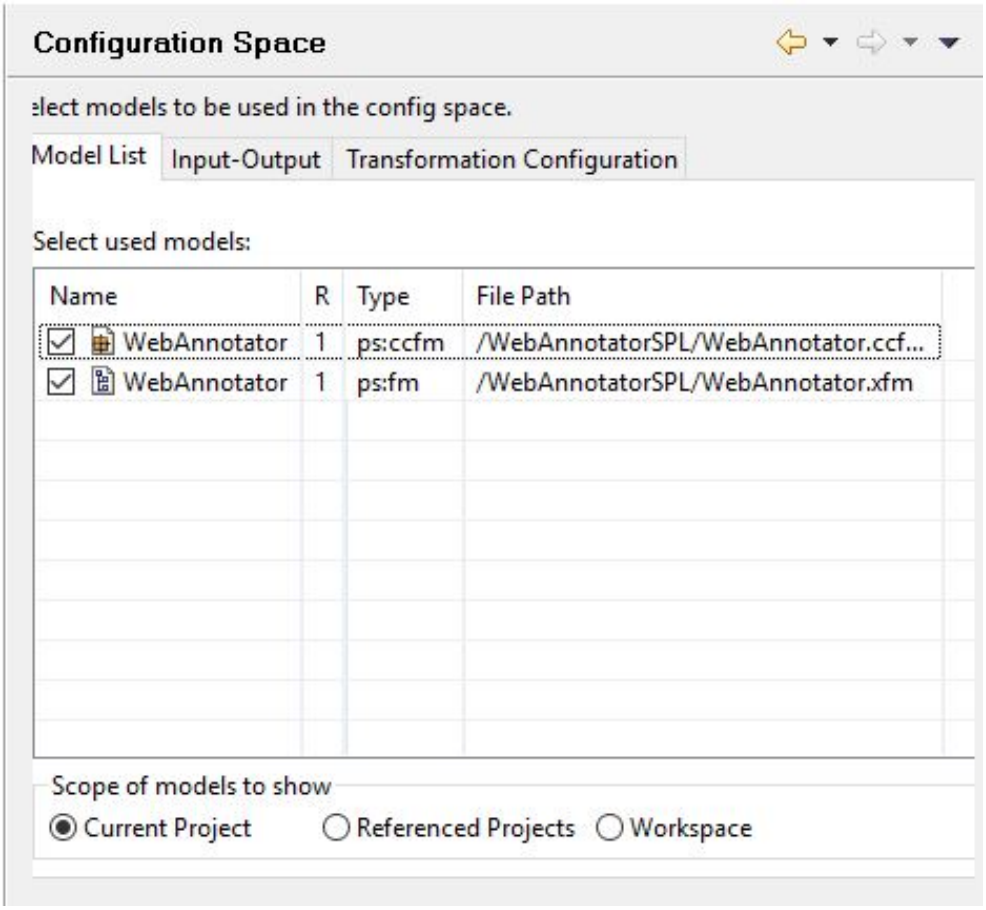


Figura 7.6: Configuración de la lista de modelos del *Configuration Space* del proyecto.

En la configuración también es necesario establecer cuál es la carpeta del código fuente (input) y la carpeta donde se deben almacenar lo productos generados (output).

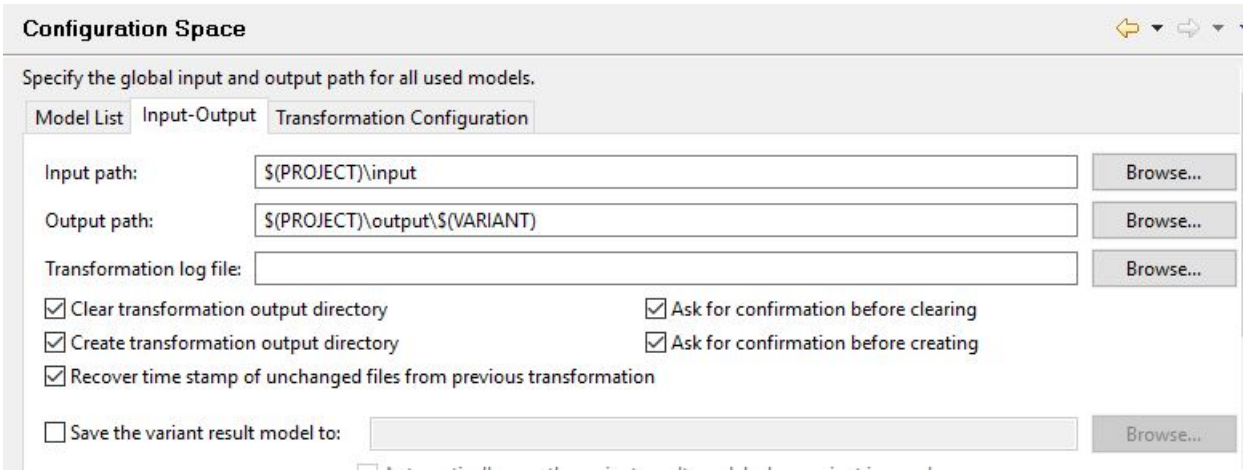


Figura 7.7: Configuración de los Input-Output del *Configuration Space* del proyecto.

Para finalizar la configuración del *Configuration Space* hay que definir los módulos que recogen y ejecutan las acciones de transformación de los modelos VDM. Los módulos que se utilizan en el espacio de configuración del proyecto son los de la transformación estándar.

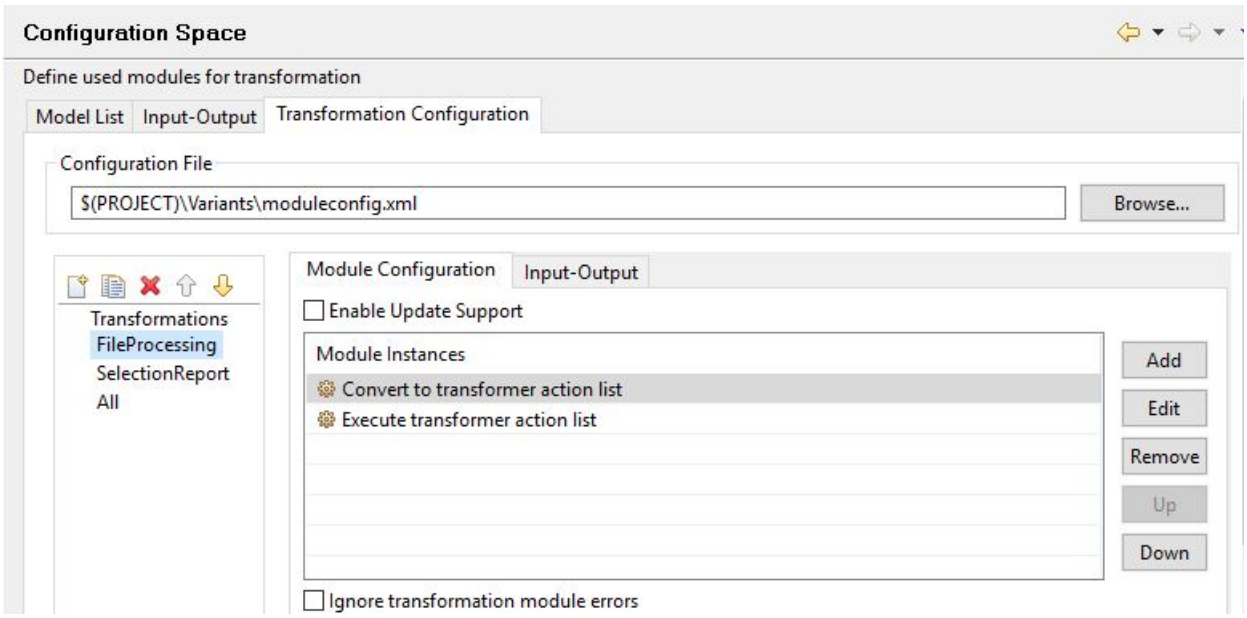


Figura 7.8: Configuración de las transformaciones VDM del *Configuration Space* del proyecto.

Una vez configurado el espacio de trabajo, ya se pueden crear modelos VDM dentro del *Configuration Space* para la configuración de productos diferentes.

7.2. Implementación de *Features*

Al igual que para preparar el entorno de trabajo, para realizar la implementación de las *Features* se han tenido que realizar varias actividades 7.9. En este caso, salvo la implementación de las primeras *Features* de *Review&Go*, el resto de *Features* se ha implementado de manera iterativa.

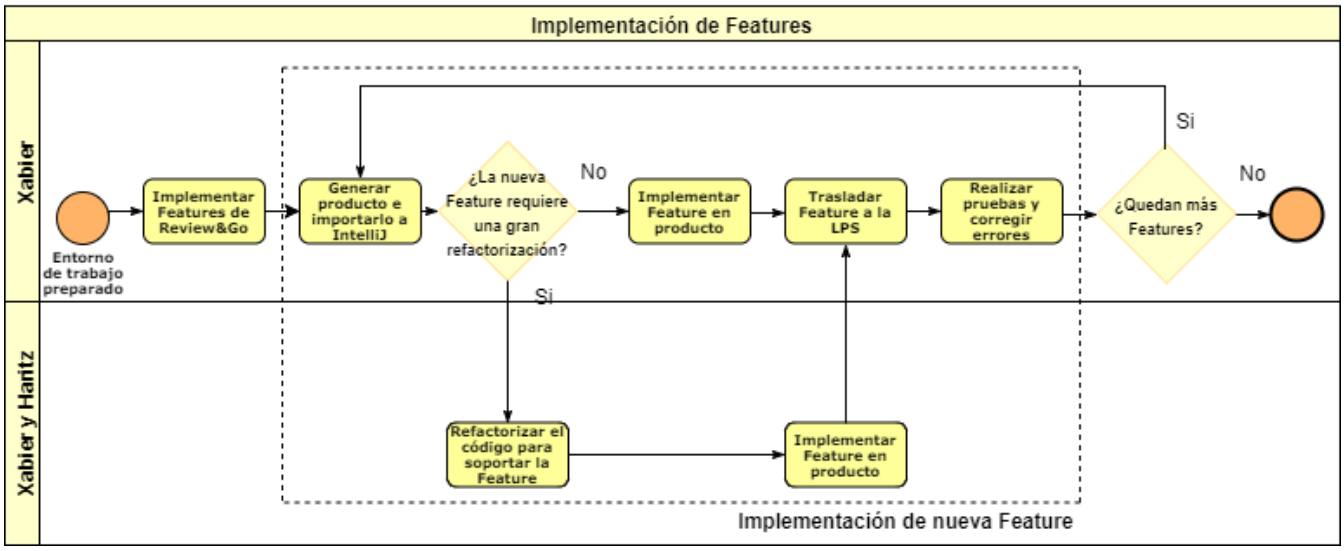


Figura 7.9: Actividades a realizar para preparar el entorno de trabajo.

Para comenzar a implementar las *Features*, se han comenzado a extraer las *Features* embebidas en el código del producto *Review&Go*. Las primeras características implementadas han sido las que pertenecen al *toolset* de la aplicación: Canvas, TextSummary, DeleteGroup, Screenshot y LastAnnotation.

El *toolset* es un elemento en el que se encuentran varias funcionalidades de la extensión. En los productos de partida el *toolset* se encuentra en *Mark&Go* y *Review&Go*, pero el código de las dos aplicaciones se implementa de manera diferente. En las dos herramientas para construir el *toolset* se utiliza un procedimiento y ficheros diferentes, por lo tanto, para la implementación de las *Features* que pertenecen a la caja de herramientas se han refactorizado estos ficheros para implementar todas estas funcionalidades en un mismo fichero, en el que se han etiquetado las *Features* que se encuentran en él.

Estas han sido las cinco primeras *Features* de la LPS. Para realizar la implementación, al tratarse de características que están repartidas por los diferentes ficheros base del proyecto, se han realizado los cambios necesarios a través del entorno IntelliJ IDEA sobre los ficheros de la carpeta input, y al ver que las cuatro *Features* funcionaban correctamente con la nueva implementación se ha procedido a etiquetar el código del fichero correspondiente al *toolset* en la LPS y se han relacionado los ficheros que recogen las funcionalidades de las *Features* con su *Feature* correspondiente en el *Family Model*.

```
// PVSCL:IFCOND(Canvas,LINE)
// Set Canvas image
let canvasImageUrl = chrome.extension.getURL('/images/overview.png')
this.canvasImage = $(toolsetButtonTemplate.content.firstElementChild).clone().get(0)
this.canvasImage.src = canvasImageUrl
this.canvasImage.title = 'Generate canvas' // TODO i18n
this.toolsetBody.appendChild(this.canvasImage)
this.canvasImage.addEventListener('click', () => {
  this.canvasButtonHandler()
})
// PVSCL:ENDCOND
// PVSCL:IFCOND(TextSummary,LINE)
// Set TextSummary image
let textSummaryImageUrl = chrome.extension.getURL('/images/generator.png')
this.textSummaryImage = $(toolsetButtonTemplate.content.firstElementChild).clone().get(0)
this.textSummaryImage.src = textSummaryImageUrl
this.textSummaryImage.title = 'Generate review report' // TODO i18n
this.toolsetBody.appendChild(this.textSummaryImage)
this.textSummaryImage.addEventListener('click', () => {
  this.textSummaryButtonHandler()
})
// PVSCL:ENDCOND
// PVSCL:IFCOND(DeleteGroup,LINE)
// Set DeleteGroup image
let deleteGroupImageUrl = chrome.extension.getURL('/images/deleteAnnotations.png')
this.deleteGroupImage = $(toolsetButtonTemplate.content.firstElementChild).clone().get(0)
this.deleteGroupImage.src = deleteGroupImageUrl
this.deleteGroupImage.title = 'Delete all annotations in document' // TODO i18n
this.toolsetBody.appendChild(this.deleteGroupImage)
this.deleteGroupImage.addEventListener('click', () => {
  this.deleteGroupButtonHandler()
})
```

Figura 7.10: Parte del fichero Toolset.js donde se muestra código etiquetado.

A partir de este momento, todos los cambios y código que hay que incluir en el proyecto para añadir nuevas *Features* no se pueden realizar sobre los ficheros de la carpeta input, porque contiene código con etiquetas y por lo tanto, no se pueden hacer pruebas. Por lo tanto, el procedimiento para implementar las nuevas *Features* es el siguiente: se genera

un producto desde la LPS y el código se importa a un proyecto de IntelliJ para realizar los cambios necesarios y las implementaciones que se requieren hasta conseguir el correcto funcionamiento de la *Feature*.

Una vez se consigue que la *Feature* haga la función que le corresponde correctamente, se trasladan los cambios realizados a la LPS. Que una *Feature* haga su función correctamente significa que la implementación realizada sobre el producto generado le ha añadido la funcionalidad o las características que reunía la nueva *Feature*.

Para realizar los cambios en la LPS se ha utilizado nuevamente DiffMerge 2.6.4, la herramienta ha permitido ver el código que se ha añadido al proyecto comparando el producto generado sin aplicarle los cambios y el mismo producto, pero con la nueva *Feature* implementada.

En el caso de que la nueva *Feature* contenga código embebido dentro de algún fichero, se etiqueta utilizando el mecanismo adecuado de *pure::variants*. Hay que recordar que si el fichero contiene etiquetas de *pure::variants* para implementar las *Features* es necesario indicar en el *Family Model* que el tipo del fichero es ps:pvscltext o ps:pvsclxml.

En el caso de que la nueva *Feature* provoque incluir nuevos ficheros, se debe actualizar el *Family Model* para que haga el mapeo de los nuevos ficheros con elementos del modelo y finalmente, se relacionan con la *Feature* correspondiente. En el caso de ser necesario, se le añade alguna restricción.

Una vez se ha implementado la nueva *Feature* en la LPS se han realizado pruebas para comprobar su correcto funcionamiento mediante la generación de diferentes productos que contengan esta *Feature*.

Este proceso se va repitiendo para cada *Feature* de tal forma que paulatinamente se añaden nuevas *Features* y se suma variabilidad a la LPS. Durante este proceso se han encontrado *Features* que se han implementado fácilmente etiquetando partes de código o sumando código de los demás productos de partida sin tener que realizar grandes modificaciones. Pero al igual que ha habido *Features* sencillas, también ha habido algunas complejas que han requerido cambios en la implementación o en su funcionalidad.

Entre las *Features* que se han terminado de implementar está la *Feature* Local. Esta *Feature* no estaba implementada en ninguno de los productos de partida, pero a lo largo del proyecto, se ha cambiado el sistema de almacenaje externo de *Review&Go* por uno Local. Esto ha permitido que se haya podido añadir esta nueva *Feature* y tener más variedad a la hora de almacenar las anotaciones.

Para realizar la implementación de *Features* que han requerido grandes cambios o incluso, alguna *Feature* que aún no estaba implementada en los productos de partida (es el caso de la *Feature* Local) se ha contado con la colaboración de Haritz a través de reuniones. A continuación, se explicarán algunas de las *Features* que han requerido grandes cambios y en las que se han tenido que aplicar refactorizaciones.

7.3. Refactorización de *Features*

7.3.1. Nuevo modelo de datos

Una de las grandes refactorizaciones se ha realizado en la parte de crear los botones del *highlighter*. En los tres productos, los botones del contenedor de botones se generaban de manera muy distinta y esto afectaba directamente a las *Features* User, GSheetProvider y MoodleProvider, que son las encargadas de crear las etiquetas que se utilizan para la generación de los botones para realizar anotaciones.

Para la generación de los botones del contenedor se ha implementado un modelo de datos que representa los niveles de clasificación Themes y Codes.

Para remplazar los ficheros que formaban el anterior modelo de datos, se ha creado un nuevo modelo de tres niveles: AnnotationGuide (se almacenan metadatos de las anotaciones y guardan los theme del grupo de anotaciones), Theme (representa un botón del tipo theme) y Code (representa un botón del tipo code).

En la figura 7.11 se muestran los elementos que componen el modelo de datos, la relación que hay entre ellos y los atributos y métodos que utilizan para la generación de los botones del *highlighter*.

Aparte de los atributos y métodos que contienen los elementos del modelo de datos general, pueden contener algún atributo o método extra si alguna *Feature* lo requiere. El AnnotationGuide a parte de contener los Themes del grupo de anotaciones también puede contener metadatos necesarios como atributos. Por ejemplo, si el modelo de datos se crea a través de GSheetProvider (genera las etiquetas a través de una hoja de cálculo de Google) en el AnnotationGuide a través de un atributo extra se guarda el identificador de la hoja de cálculo que ha generado las anotaciones de las etiquetas.

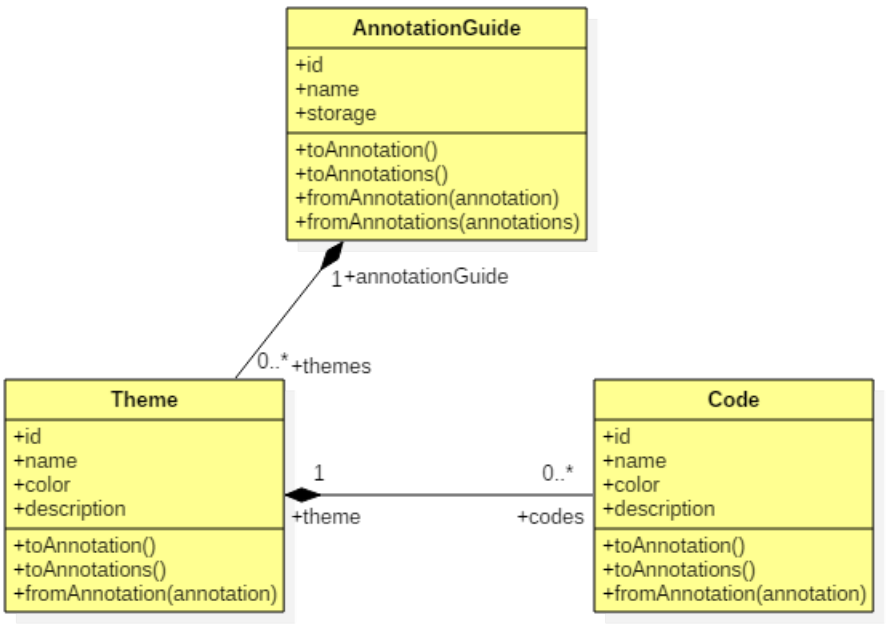


Figura 7.11: Nuevo modelo de datos para realizar anotaciones.

La idea del modelo de datos es ayudar a realizar el proceso de crear los Theme y Codes (si se utilizan en la extensión) para generar los botones del *highlighter*. Primero, por medio de un método que se crea en el AnnotationGuide para cada medio que proporciona la información requerida para generar las etiquetas de los botones, se crea el modelo de datos por primera vez. En segundo lugar, a través del modelo de datos se crean las anotaciones a través de los métodos toAnnotation() y toAnnotations(). Finalmente, cuando en la aplicación llega el momento de crear los botones del *highlighter*, se recogen las anotaciones del almacén de anotaciones y se vuelve a generar el modelo de datos a través de los métodos fromAnnotation() y fromAnnotations(). Una vez se recupera el modelo de datos, se genera los botones con un método nuevo que crea los botones de la misma manera para todos los casos.

7.3.2. Añadir botones dinámicamente (Dynamic)

En *Review&Go* existe la posibilidad de añadir nuevos botones personalizados al *highlighter* de manera dinámica, pero su implementación no es nada genérica y es muy dependiente, porque el botón de crear nuevos botones solo se encuentra dentro de un Theme de la configuración predefinida (en el Theme Other). Por lo tanto, se ha realizado la refacto-

rización de esta *Feature* para que el botón de crear nuevos Themes sea un botón más del *highlighter* y no sea dependiente. Además, como extra, a esta *Feature* se le ha implementado que realizando click derecho en un Theme se puedan crear también nuevos botones dentro. Esta funcionalidad para esta *Feature* se ha sacado de una de las actualizaciones que se ha realizado en *Highlight&Go* a mediados de abril por parte de Haritz.

7.3.3. Generación de hoja de cálculo (GSheetConsumer)

En *Highlight&Go* los botones se generan a través de una hoja de cálculo y al realizar las anotaciones en los recursos, las anotaciones se trasladan a la hoja de cálculo porque se encuentra sincronizada. A lo largo del transcurso del proyecto, el producto *Highlight&Go* ha incorporado grandes modificaciones por parte de Haritz y entre ellas, una es la manera de consumir las anotaciones en la hoja de cálculo.

En la nueva versión de *Highlight&Go* en vez de tener la hoja de cálculo de origen sincronizada con la extensión, existe la opción de generar una hoja de cálculo nueva a través de un botón del *toolset*. Por lo tanto, esta manera de consumir anotaciones se ha considerado más genérica y se le ha visto más sentido dentro de una LPS, porque permite que los botones no se hayan creado necesariamente a partir de una hoja de cálculo, ya que las nuevas hojas se crean independientemente.

7.3.4. Validar una anotación (Validate)

Al igual que la *Feature* GSheetConsumer, la opción de validar una anotación también se trata de una característica de *Highlight&Go*. Por lo tanto, también se han realizado cambios en la funcionalidad de esta *Feature*.

Validar una anotación consiste en dar una respuesta a una de las anotaciones verificando que su contenido es correcto. Al tratarse de una respuesta a una anotación y contar con una *Feature* llamada Reply cuyo fin también es permitir a los usuarios responder a una anotación, se ha encontrado un solapamiento en la funcionalidad.

Anteriormente, una anotación se validaba haciendo click derecho sobre ella y eligiendo la opción Validate dentro del *context menu* que se abría. Ahora esta funcionalidad, es más completa. Al elegir la opción de validar la anotación al usuario se abre una nueva ventana y al igual que en la *Feature* de Reply, se muestran todas las respuestas que ha recibido la anotación. Además, existe la opción de introducir una razón por la que se ha realizado

la validación, como si fuera una respuesta. Cuando se realiza una validación la razón por la que se ha validado aparecerá entre las respuestas que ha recibido la anotación, pero aparecerá subrayada en verde para distinguirla entre las respuestas de los usuarios.

Por lo tanto, como en la interfaz de realizar la validación es lógico que se muestren las respuestas de los demás usuarios para considerar la validación, se ha decidido que la *Feature* Validate solo pueda ser seleccionada cuando se dispone de la *Feature* Reply, estableciendo una restricción en el *Feature Model*.

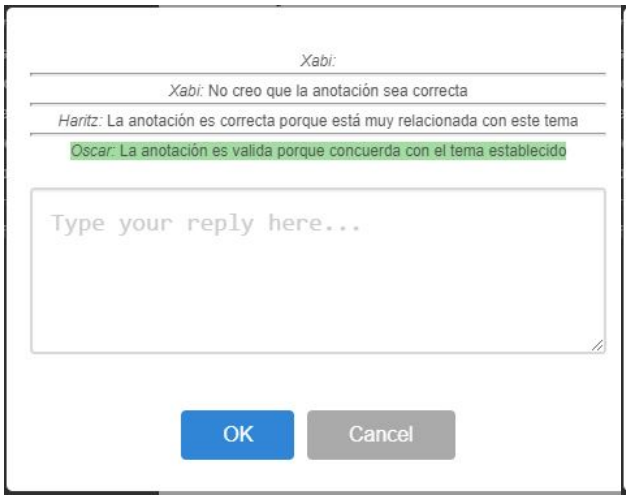


Figura 7.12: Implementación para la *Feature* Reply.

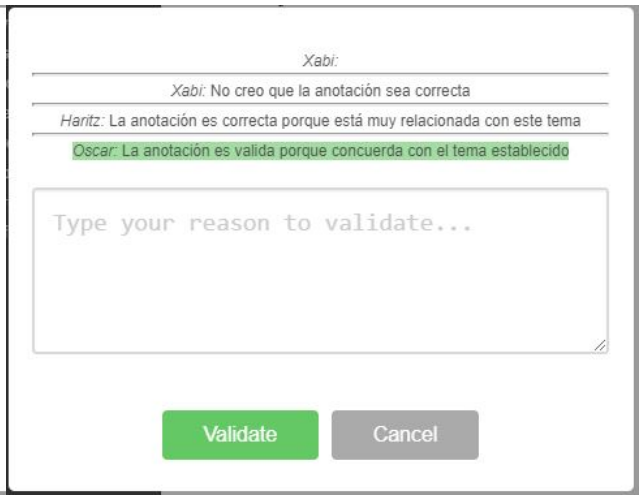


Figura 7.13: Implementación para la Feature Validate.

En las figuras 7.12 y 7.13 se muestran las ventanas que se abren al realizar una respuesta y una validación respectivamente. Entre las respuestas que se han realizado, se encuentra una subrayada en verde. Esta respuesta se trata de una validación realizada por el usuario Óscar a la anotación correspondiente.

7.4. Obtención de productos a partir de *Features*

Uno de los propósitos principales por el que se ha realizado el proyecto es poder mantener los productos por medio de la LPS. Por lo tanto, al finalizar la implementación de las *Features* es importante comprobar que se pueden reconstruir los productos base a partir de las *Features* de la LPS.

Con las *Features* que se han podido implementar dentro del tiempo disponible, se pueden reconstruir dos de los productos: *Highlight&Go* y *Review&Go*. *Mark&Go* no se puede

reconstruir del todo porque han quedado sin implementar las siguientes *Features*: MoodleProvider, MoodleConsumer y ResouceBased.

A pesar de no tener la posibilidad de reconstruir *Mark&Go*, es posible incluir a los productos que se generan, algunas funcionalidades que eran propias de *Mark&Go*. Por ejemplo, se ha extraído la *Feature* Reply porque puede ser interesante para más productos.

El producto de *Highlight&Go* se reconstruye a partir de las *Features* de la figura 7.14.

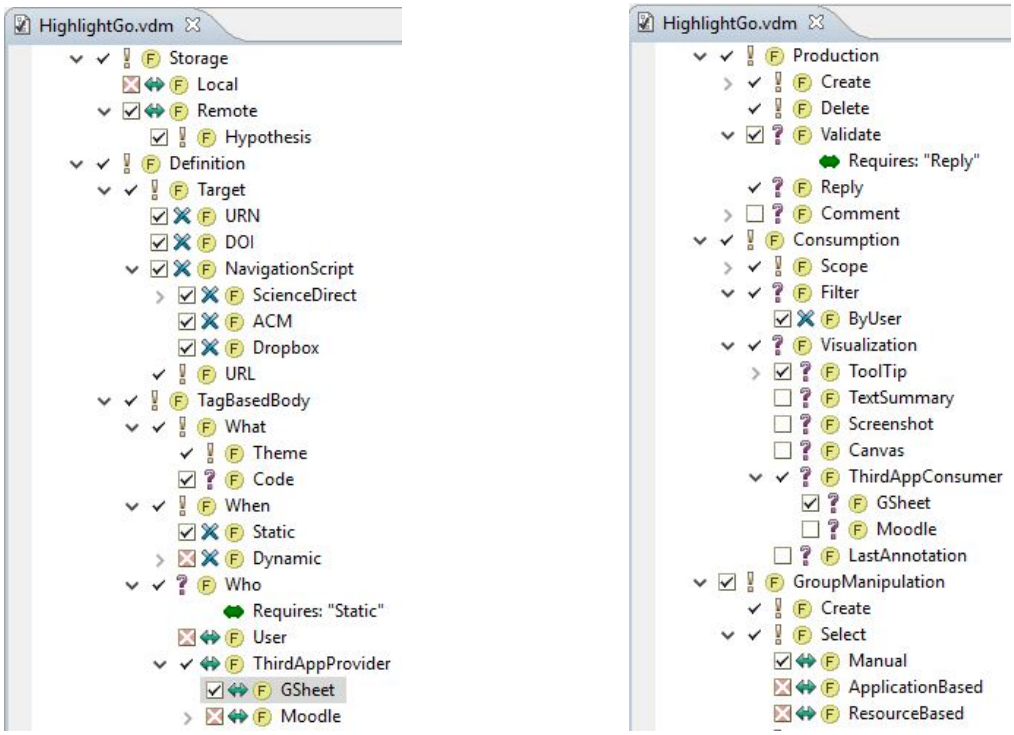


Figura 7.14: *Features* seleccionadas para reconstruir *Highlight&Go*.

A través del producto generado con el VDM anterior, los botones del *highlighter* se crean a partir de una hoja de cálculo. Se permiten validar y dar respuesta a anotaciones. El grupo de anotaciones que se almacena en *Hypothes.is* se selecciona de manera manual, y una vez generadas las anotaciones, se puede generar una hoja de cálculo en la que se visualizan las anotaciones realizadas. Además, para visualizar las anotaciones, se dispone de un filtro que muestra las anotaciones creadas por los usuarios elegidos.

Por otro lado, para comprobar que el producto de *Review&Go* se reconstruye totalmente, se han seleccionado las *Features* de la figura 7.15.

Mediante el VDM de la figura 7.15, se obtiene un producto similar a *Review&Go*. Los botones del *highlighter* se crean a partir una configuración preestablecida por el usuario. Se permiten comentar las anotaciones y complementarlas con diferentes opciones.

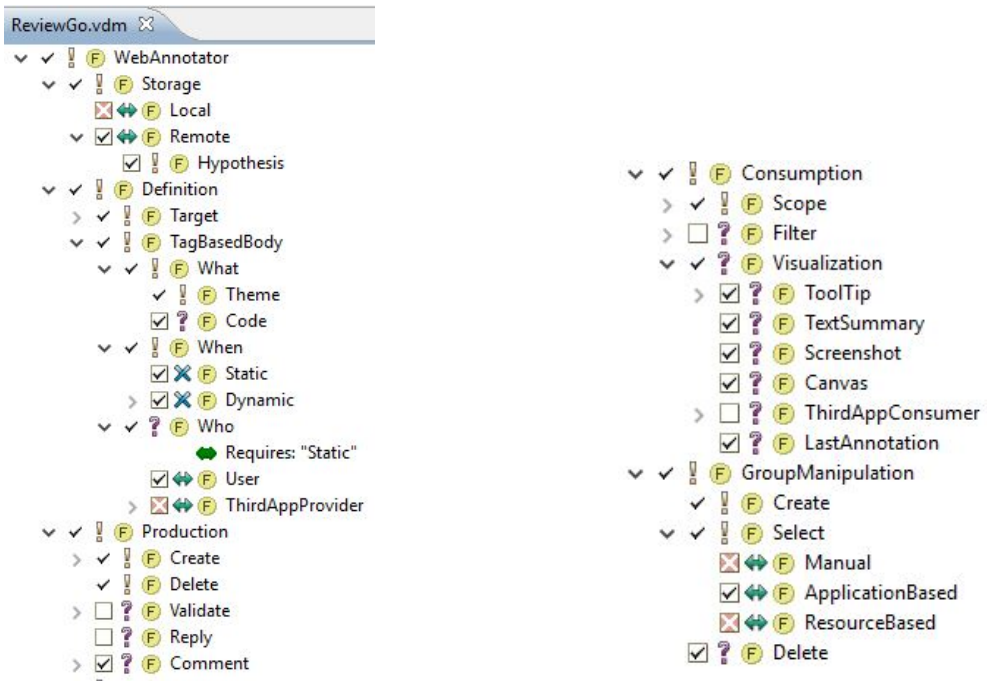


Figura 7.15: *Features* seleccionadas para reconstruir *Review&Go*.

El grupo de anotaciones que se almacena en *Hypothes.is* se selecciona de manera automática por la aplicación, y una vez generadas las anotaciones, se pueden realizar diferentes visualizaciones a través de las funcionalidades que se encuentran en el *toolset*. Además, es posible eliminar todas las anotaciones del grupo gracias a que la *Feature* DeleteGroup esta incorporada en el producto.

A parte de estos productos, gracias a que las *Features* de la LPS se han realizado de forma genérica, es posible crear una gran cantidad de productos diferentes realizando diferentes combinaciones entre las *Features* disponibles.

7.5. Resultados del proyecto

En este apartado se recogen los resultados que se han obtenido tras haber dado por finalizada la implementación de *Features*. El proceso de realizar las implementaciones ha sido largo. Las refactorizaciones han llevado a cabo mucho tiempo, pero finalmente se ha conseguido implementar la gran mayoría de *Features* que estaban establecidas en el *Feature Model* inicial y mediante la LPS se pueden generar una gran cantidad de productos.

En las siguientes tablas de este último apartado se muestran las *Features* que se han implementado. Las tablas están divididas por los diferentes apartados que recogen grupos

de *Features* de la LPS y contienen los ficheros que están relacionados con cada *Feature*. En cada fichero relacionado con la *Feature*, se indica que tipo de dependencia tiene con la característica, en la que se encuentran dos posibilidades: una posibilidad es que el fichero contenga parte del código de la *Feature* dentro (C) y la otra posibilidad es que el fichero sea solamente dependiente de esa *Feature* (F).

7.5.1. Storage

En primer lugar, se encuentra el apartado de las *Features* de Storage. Tal y como se ha mencionado anteriormente, los productos de partida solo contaban con la opción de almacenamiento Hypothes.is, pero durante el transcurso del proyecto en una de las actualizaciones de uno de los productos se ha implementado el almacenamiento local. Al realizar el *Feature Model* 6.1, la opción de almacenar las anotaciones localmente en el navegador ya estaba contemplada como posible futura alternativa de almacenamiento en la *Feature* llamada Local.

Por lo tanto, en los productos que se generan a través de la LPS, es posible elegir almacenar las anotaciones localmente en el navegador (Local) o remotamente con *Hypothes.is*.

Grupo de Feature	**Feature**	**Ficheros**	**Tipo de dependencia**
Storage	Remote/ Hypothesis	Hypothesis.js	F
		HypothesisClientManager.js	F
		ContentScriptManager.js	C
		scienceDirectContentScript.js	C
		GoogleSheetContentScriptManager.js	C
	Local	EmptyDatabase.json	F
		LocalStorageClient.js	F
		LocalStorageManager.js	F
		ContentScriptManager.js	C
		scienceDirectContentScript.js	C
		GoogleSheetContentScriptManager.js	C

Tabla 7.1: Composición de las *Features* que se agrupan en Storage.

En la tabla 7.1, se encuentras las *Features* que se encuentran dentro de la *Feature* Storage. Los ficheros en los que se etiqueta código son en los que se crea el objeto de almacenamiento, porque depende de la opción que se ha seleccionado en la variable, se guarda un

cliente de Hypothes.is o uno para guardar las anotaciones localmente. Además, cada *Feature* contiene ficheros exclusivos en los que se encuentran las funciones para comunicarse con la extensión.

7.5.2. Definition

Grupo de Feature	Feature	Ficheros	Tipo de dependencia
Definition/Target	URN	filePermission.html	F
		ContentTypeManager.js	C
		background.js	C
		manifest.json	C
	DOI	background.js	C
		TargetManager.js	C
		ContentTypeManager.js	C
		manifest.json	C
	ScienceDirect	scienceDirectContentScript.js	F
		TargetManager.js	C
	ACM	acmContentScript.js	F
		manifest.json	C
	Dropbox	contentScript.js	C
		TargetManager.js	C
		ContentTypeManager.js	C
		manifest.json	C

Tabla 7.2: Composición de las *Features* que se agrupan en Target

En la tabla 7.2 se encuentras las *Features* que se encuentran dentro de las características relacionadas con el Target. En la tabla se puede observar que la mayoría de *Features* se encuentra entre los ficheros ContentTypeManager.js, background.js, TargetManager.js y manifest.json.

Grupo de Feature	Feature	Ficheros	Dependencia
	What/ Code	TagManager.js	C
		TextAnnotator.js	C
		Theme.js	C
		AnnotationGuide.js	C

		UserDefinedHighlighterDefinition.json	C
		Code.js	F
	When/ Dynamic	TagManager.js.js	C
		Theme.js	C
		AnnotationGuide.js	C
	Who/ User	GroupSelector.js	C
		TagManager.js	C
		Config.js	C
		AnnotationGuide.js y Theme.js	C
		DefaultHighlighterGenerator.js	F
		UserDefinedHighlighterDefinition.json	F
	Who/ GSheetProvider	background.js	C
		Config.js	C
		AnnotationGuide.js	C
		Theme.js	C
		TextAnnotator.js	C
		GoogleSheetsManager.js	C
		manifest.json	C
		GoogleSheetClient	C
		GoogleSheetClientManager	F
		GoogleSheetContentScriptManager	F
		GoogleSheetParser	F
		HypothesisGroupInitializer	F
	Who/ MoodleProvider	No se ha terminado de implementar	-

Tabla 7.3: Composición de las *Features* que se agrupan en Tag-based body.

En la tabla anterior 7.3 se muestra en qué ficheros se han implementado las *Features* del Tag-based body que tenían parte de implementación dentro. La mayor parte de las *Features* dentro de este grupo tienen parte de código en los ficheros que representan el modelo de datos de la herramienta.

7.5.3. Production

Grupo de Feature	Feature	Ficheros	Tipo de dependencia
Production	Reply	TextAnnotator.js	C
		ReplyAnnotation.js	F
	Validate	TextAnnotator.js	C
		ReplyAnnotation.js	C
		textAnnotator.scss	C
	Comment	TextAnnotator.js	C
	Comment/ AddReference	-	Aún no esta implementada
	Comment/ Categorize	-	Aún no esta implementada
	Comment/ SentimentAnalysis	TextAnnotator.js	C
	Comment/ Autofill	TextAnnotator.js	C

Tabla 7.4: Composición de las *Features* que se agrupan en Production.

Las *Features* agrupadas dentro de Production contienen gran parte de código en el fichero TextAnnotator.js ya que contiene la mayor parte de las operaciones que se realizan sobre las anotaciones.

7.5.4. Consumption

Grupo de Feature	Feature	Ficheros	Tipo de dependencia
Consumption	Visualization/ ToolTip	TextAnnotator.js	C
	Visualization/ TextSummary	TextSummary.js	F
		Toolset.js	C
		generator.png	F
	Visualization/ Screenshot	Screenshot.js	F
		Toolset.js	C
		screenshot.png	F
	Visualization/ Canvas	Canvas.js	F
		Toolset.js	C
		overviwe.png	F
		reviewCanvas.html	F
	Visualization/ LastAnnotation	Resume.js	F
		Toolset.js	C
		resume.png	F
	Visualization/ GSheetConsumer	Toolset.js	C
		GoogleSheetsManager.js	C
		GoogleSheetClient.js	C
		googleSheet.svg	F
		GoogleSheetGenerator.js	F
	Visualization/ MoodleConsumer	Aún no esta implementada	-
	Filter/ ByUser	userFilterWrapper.html	F
		UserFilter.js	F
		ContentScriptManager.js	C
		TextAnnotator.js	C

Tabla 7.5: Composición de las *Features* que se agrupan en Consumption.

En esta tabla 7.5, se encuentran las *Features* que se agrupan dentro del grupo de caracteristicas de Consumption. De momento, las *Features* que se encuentran dentro del grupo de caracteristicas de Scope no tienen implementación en el proyecto. En la actual implementación las *Features* de Visualization no se dispone de ninguna que tenga implementada

más de un Scope, y por lo tanto tendrían la misma implementación que tienen las de Visualization. En un futuro sí que se espera que una *Feature* de visualización se pueda realizar sobre más de un Scope y, por lo tanto, se han dejado las *Features* disponibles y etiquetadas con el código de las visualizaciones.

Las *Features* que se encuentran dentro del grupo de Visualization tienen unos ficheros parecidos (código etiquetado en el Toolset.js, una imagen y un fichero específico), porque la mayoria son parte del *toolset*.

7.5.5. GroupManipulation

Grupo de Feature	Feature	Ficheros	Tipo de dependencia
GroupManipulation	Select/ Manual	manifest.json	C
		groupSelection.html	C
		ContentScriptManager.js	C
		GroupSelector.js	C
		AnnotationGuide.js	C
		HypothesisGrupInitializer.js	C
		Toolset.js	C
	Select/ ApplicationBased	GroupSelector.js	C
		AnnotationGuide.js	C
		HypothesisGroupInitializer.js	C
	Select/ ResourceBased	Aún no esta implementada	-
	DeleteGroup	TextAnnotator.js	C
		Toolset.js	C
		Events.js	C
		deleteAnnotations.png	F
		DeleteGroup.js	F

Tabla 7.6: Composición de las *Features* que se agrupan en GroupManipulation.

En la tabla 7.6 se muestra en qué lugares se encuentra la implementación de los *Features* relacionadas con el GroupManipulation. En ella se encuentran las *Features* que se encargan de realizar la selección de grupo para anotar, estas *Features* encuentran gran parte de la implementación en el fichero GroupSelector.js. Por otro lado, se encuentra la

Feature encargada de borrar las anotaciones del grupo. Esta *Feature* tiene un fichero correspondiente para realizar su función y contiene código en el fichero Toolset.js porque la funcionalidad se encuentra entre uno de los botones del *toolset*.

En la siguiente tabla 7.7 se muestran las restricciones que se han añadido al *Feature Model* para resolver conflictos en las aplicaciones y establecer requisitos de elección a la hora de generar los productos.

Restricción	**Motivo**
ScienceDirect **Requires** DOI.	La feature ScienceDirect se encarga de guardar el DOI para que no se pierda en la redirección. Por lo tanto, la aplicación debe poder gestionar identificadores DOI.
Dynamic **Conflicts** ThirdAppProvider	Si los botones se generan por una aplicación externa no tiene sentido que se puedan crear botones dinámicamente.
Who **Requires** Static	Para poder elegir que las etiquetas de los botones se creen mediante un proveedor de etiquetas al configurar la herramienta, se debe elegir que las etiquetas se deben crear de manera estática.
MoodleProvider **Requires** ResourceBased	Los grupos de Moodle requieren que sus nombres no sean identificables por la ley de protección de datos, por lo tanto, el grupo que se abra para anotar se debe elegir mediante ResourceBase que resuelve el hash del nombre del grupo.
Validate **Requires** Reply	Validate al tratarse de un tipo de respuesta a una anotación requiere que se puedan realizar respuestas a través de Reply.
Local **Conflicts** ByUser	Con el almacenamiento local, no es útil el filtro de usuarios porque este sistema de almacenamiento no permite tener más de un usuario.
ToolTip **Requires** SingleAnnotation	ToolTip está implementado para un ámbito de aplicación de SingleAnnotation.
TextSummary **Requires** ResourceAnnotationCluster	TextSummary está implementado para un ámbito de aplicación de ResourceAnnotationCluster.
Screenshot **Requires** ResourceAnnotationCluster	Screenshot está implementado para un ámbito de aplicación de ResourceAnnotationCluster.

Canvas **Requires** ResourceAnnotationCluster	Canvas está implementado para un ámbito de aplicación de ResourceAnnotationCluster.
GoogleConsumer **Requires** AnnotationGroup	GoogleConsumer está implementado para un ámbito de aplicación de AnnotationGroup.
MoodleConsumer **Requires** ResourceAnnotationCluster	MoodleConsumer está implementado para un ámbito de aplicación de ResourceAnnotationCluster.

Tabla 7.7: Restricciones establecidas en el *Feature Model*.

En este punto queda acaba la LPS para el proyecto. Se ha conseguido implementar la gran mayoría de *Features* establecidas en el *Feature Model*. Las *Features* que no se han podido implementar por falta de tiempo son siguientes: MoodleProvider, MoodleConsumer, AddReference, Categorize y ResourceBased.

Con todas las *Features* que se han implementado es posible generar de nuevo un *Review&Go* y una herramienta parecida a *Highlight&Go*, pero además, es posible añadirles nuevas funcionalidades gracias a la generalización con las que se han implementado las *Features*. Por ejemplo, a través de la LPS es posible crear una herramienta como *Review&Go*, pero que permita trasladar las anotaciones a una hoja de cálculo y realizar respuestas a las anotaciones.

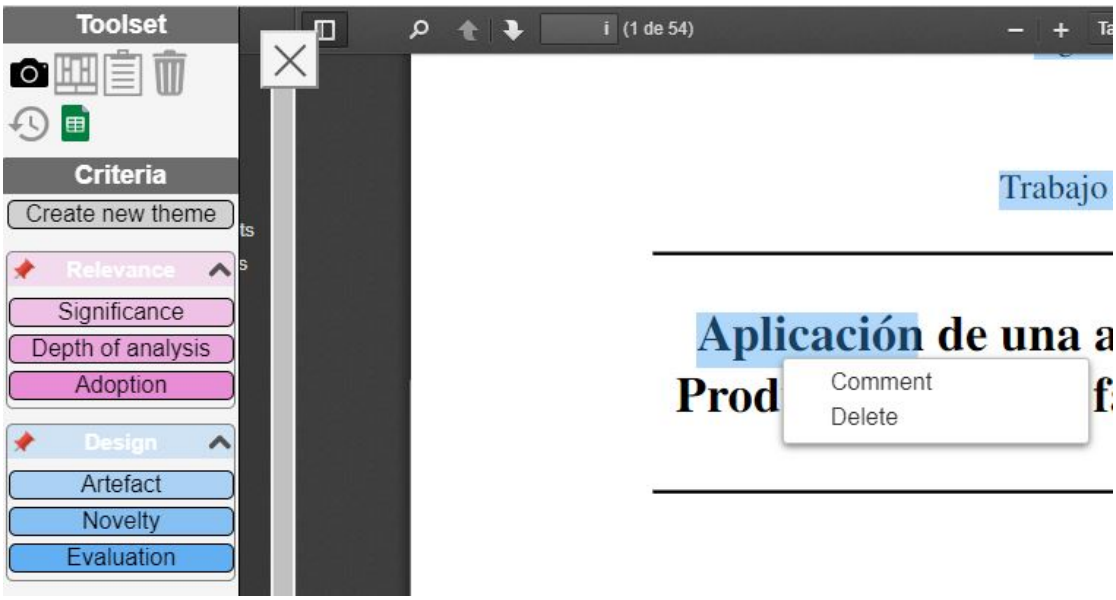


Figura 7.16: Producto generado a través de la LPS.

En un futuro se espera que se terminen de implementar las *Features* que quedan con más tiempo y poder generar extensiones parecidas a *Mark&Go*. Además, esta LPS dará soporte a futuras implementaciones que ayuden a completar los productos que se pueden generar y ayudar a cumplir más requisitos.

Para concluir, la fase de implementación ha sido un proceso duro en el que se han tenido que realizar abundantes cambios en el código de origen. Para realizar los cambios ha sido necesario manejarse bien con JavaScript, lo que ha aportado nuevos conocimientos. La fase se ha terminado de manera satisfactoria porque se han conseguido implementar más *Features* de las esperadas con éxito.

8. CAPÍTULO

Seguimiento del proyecto

En este capítulo se muestra el seguimiento y control realizado del proyecto. Se estudia la planificación inicial comparándola con lo que realmente ha acontecido en el proyecto. En primer lugar, se ha realizado una comparación de las horas que se estimaban hacer en cada tarea con las que realmente se han necesitado. En segundo lugar, se explicarán las actividades que se han realizado para llevar a cabo el seguimiento del proyecto. En tercer lugar, se comentarán los riesgos que han surgido y cómo se ha pretendido gestionar la calidad del proyecto. Por último, se muestra cuál ha sido el diagrama Gantt final.

8.1. Dedicación al proyecto

Al finalizar el proyecto, se han recogido las horas invertidas en el proyecto para realizar una comparación en la siguiente tabla 8.1 de las horas trabajadas realmente, con las que inicialmente estaban estimadas.

Durante el desarrollo del proyecto se tuvieron que realizar algunos cambios sobre la planificación inicial. Las tareas de desarrollo y las pruebas del proyecto se esperaban realizarlas en dos fases. En la primera fase se iba a realizar el proceso de etiquetado, la implementación y las pruebas con solo dos de los productos: *Highlight&Go* y *Review&Go*. En cambio, en la segunda fase, estaba planeado realizar las mismas tareas, pero con *Mark&Go* para completar la LPS.

Al comenzar a analizar y etiquetar el código se vio que era mejor realizar las actividades

sobre los tres productos en una misma fase. Durante el análisis de código de los dos primeros productos, se encontraron varios ficheros o partes de código que pertenecían a *Mark&Go* lo que resultaba bastante confuso a la hora de etiquetar el código. Por otro lado, también se pensó que realizar las actividades en dos fases llevaría mucho más tiempo y que sería mejor etiquetar el código de los tres productos, conseguir el *Feature Model* completo e ir implementando las *Features* libremente.

Tarea	Dedicación estimada	Estimación real	Diferencia
Formación			
Refrescar LPS	3 h	4 h	+1
Pure::variants	30 h	8 h	-22
Refrescar JavaScript	5 h	3 h	-2
Herramientas de anotaciones web	24 h	15 h	-9
Desarrollo			
Proceso de etiquetado I	35 h	67 h	+32
Implementación *pure::variants* I	25 h	72 h	+47
Proceso de etiquetado II	30 h	-	-30
Implementación *pure::variants* II	20 h	-	-20
Pruebas			
Testeo I	17 h	41 h	+24
Testeo II	18 h	-	-18
Validación y correcciones finales	10 h	18 h	+8
Gestión			
Definición de objetivos	1 h	1 h	0
Reuniones con el tutor	10 h	3 h	-7
Reuniones con el desarrollador de las herramientas	10 h	18 h	+8
Seguimiento y control	15 h	8 h	-7
Planificación	18 h	18 h	0

Sigue en la página siguiente.

Tarea	Dedicación estimada	Estimación real	Diferencia
Documentación			
Realizar memoria	80 h	105 h	+25
Preparar defensa	20 h	20 h	0
Total	**371 h**	**401 h**	**+30**

Tabla 8.1: Comparativa entre la dedicación estimada y real.

Para hacer la comparación de la horas estimadas y reales, no se han contado las actividades que pertenecían a la segunda fase de desarrollo y pruebas. Las tareas que se han realizado finalmente en una sola fase se han contabilizado en las tareas que pertenecían a la primera fase anterior, pero se han realizado sumándoles el producto de *Mark&Go*, por lo tanto, la diferencia de la dedicación estimada y las horas invertidas realmente es grande.

La fase de formación ha requerido mucho menos tiempo de lo que estaba estimado, esto se debe sobre todo a que la formación con *pure::variants* no ha necesitado tanto tiempo como el que se esperaba. Para aprender a utilizar *pure::variants* se ha leído gran parte del manual y se han utilizado varios ejemplos que proporciona *pure::variants* para entender la herramienta. La parte práctica de la herramienta se ha aprendido a la hora de realizar la implementación. Por eso no se han requerido tantas horas en la formación con esta herramienta.

En la fase de implementación se han invertido muchas horas. En algunos casos se han tenido que realizar refactorizaciones que han necesitado bastante tiempo y además, el tiempo para realizar los testeos y correcciones también ha sido más de lo esperado porque al etiquetar el código en *pure::variants* se ha necesitado ser cuidadoso porque tiene que estar todo bien estructurado para que no se den errores.

Al realizar la implementación, realizar las pruebas de las extensiones que se generaban era una tarea que requería mucho tiempo porque el testeo se debe realizar sobre el navegador, lo que suponía generar el proyecto, instalar las dependencias, crear los ficheros de ejecución, instalar la extensión y utilizar el *debugger* de *Chrome* para encontrar los errores.

En cuanto a las tareas de gestión, no ha habido una gran diferencia de horas. Las reuniones con el tutor han sido menos de las esperadas. Las reuniones han sido breves y la gran mayoría para realizar el seguimiento del proyecto. Las reuniones más largas han sido para acordar la estructura del *Feature Model*.

En cuanto a las reuniones con Haritz, se han invertido más horas de lo previsto. La mayor parte del tiempo ha sido para aprender a utilizar las extensiones y resolver dudas sobre el código. En esta tarea no se han contabilizado las horas invertidas con Haritz en la implementación de refactorizaciones y *Features* nuevas. Estas horas se han contabilizado en la tarea de implementación.

En la tarea de seguimiento y control se han contabilizado las horas invertidas en realizar las actas de las reuniones (anexo A) y en la contabilización de las horas que ha requerido cada tarea. Además, también se ha contado el tiempo que se ha requerido para realizar las comunicaciones con el tutor: envió de correos, tareas, documentación...

Por último, una tarea que también destaca por la diferencia de tiempo que ha sido requerido es la realización de la memoria. En esta tarea se han invertido 25 horas más de lo esperado. Esto se debe a que se ha realizado una memoria extensa y se ha pretendido que el contenido sea lo más claro y entendible posible, lo que ha requerido realizar muchas revisiones.

En total se han invertido 401 horas en la realización del proyecto si finalmente se invierten 20 horas en preparar la defensa del proyecto. Se han invertido 30 horas más de las que estaban estimadas inicialmente, esto se debe a la realización de refactorizaciones que no estaban contempladas y el esfuerzo extra que ha supuesto redactar la memoria. Por otro lado, la gran cantidad de horas que se han invertido en las dos tareas mencionadas anteriormente se ven compensadas con la menor cantidad de horas que se han necesitado para la formación.

8.2. Soluciones a riesgos y gestión de calidad

En la planificación inicial se realizó una identificación de posibles riesgos que podrían surgir en el transcurso del proyecto. En este apartado se mencionan los problemas que han surgido realmente en la realización del proyecto y cómo se les ha dado solución.

- **Problemas personales:** Afortunadamente no ha habido ningún tipo de problema personal que pusiera en riesgo completar el proyecto.
- **Perdida de información:** Gracias al uso de plataformas como GitHub y Google Drive no han existido problemas de pérdidas de información. Además, utilizar GitHub ha permitido recuperar anteriores versiones de ficheros cuando ha sido necesario.

- **Avería del ordenador:** Durante el desarrollo del proyecto no ha habido ninguna avería que retrasase el proyecto. En cambio, si que ha habido problemas en cuanto al rendimiento del ordenador. Al tener que realizar ejecuciones costosas y no contar con un ordenador muy potente, algunas de las ejecuciones tardaban mucho tiempo o se formaban bloqueos por toda la actividad que había en el momento.

- **Asignaturas de cuarto curso:** Debido a la buena organización realizada para compaginar las asignaturas y el proyecto, no ha habido problemas para realizar con éxito todo el trabajo. Las asignaturas de cuarto sí que han provocado que no se le diese el tiempo suficiente al proyecto en algunos momentos. Por ejemplo, el cúmulo de trabajo y exámenes de las asignaturas en el mes de marzo y la primera semana de abril, ralentizaron las tareas del proyecto.

  Una vez finalizadas las asignaturas a mediados de abril, se han podido invertir las horas necesarias al proyecto para realizar la parte de la implementación y la memoria.

- **Problemas con la comprensión del código y necesidad de realizar refactorizaciones:** A lo largo del proyecto estos son los problemas que más han surgido. Afortunadamente, para poder solucionar estos problemas se ha contado con la ayuda de Haritz que siempre se ha mostrado dispuesto a echar una mano para resolver todo tipo de dudas.

- **Problemas con la licencia de *pure::variants*:** La licencia que se adquirió inicialmente fue una licencia mensual gratuita. Para poder seguir utilizando *pure::variants*, cada vez que se caducaba una licencia se ha solicitado una nueva, por lo tanto, las licencias se han tenido que renovar mensualmente con diferentes cuentas.

- **Problemas para entender *pure::variants*:** Utilizar *pure::variants* ha sido más fácil de lo esperado gracias a que las acciones se realizan muy intuitivamente. En la primera reunión, el exalumno Raúl Medeiros realizó una presentación muy beneficiosa de esta herramienta para explicar sus características mediante ejemplos de cómo se utiliza.

En cuanto a la calidad del proyecto, se han llevado a cabo varios esfuerzos para que los entregables tengan un buen nivel de calidad.

- **Calidad de la LPS:** Para que el proyecto tenga buena calidad se han tenido en cuenta varios aspectos. En primer lugar, se ha mantenido que el código del proyecto siga el formato que ya seguía anteriormente, concretamente el código trata

de seguir el JavaScript Standard Style[1]. En segundo lugar, se ha tratado que todas las especificaciones (*Feature Model*, comentarios...) que tiene la LPS se realicen en inglés para seguir con la línea de internacionalizar los productos. Por último, las configuraciones se han realizado de forma adecuada y se han realizado comentarios a lo largo del código que hagan el contenido más comprensible.

- **Calidad de la memoria:** Para la realización de la memoria se ha tratado de no realizar faltas ortográficas utilizando diferentes correctores y realizando múltiples revisiones. En cuanto al contenido, se han intentado realizar las explicaciones tratando de que sean entendibles para todo tipo de lectores. Para asegurar la calidad de la memoria se han revisado varias versiones anteriores por Óscar, Haritz y más gente que no controla del tema para comprobar que el contenido cuente con un buen nivel de calidad. Para evitar problemas con el formato o estructura se ha utilizado la plantilla para los TFGs de LaTeX.

8.3. Diagrama Gantt final

En esta sección se muestra cómo ha sido realmente el Gantt diagrama del proyecto. El Gantt diagrama tuvo que sufrir modificaciones a partir de marzo por no haber podido cumplir los plazos exactos y tener que modificar algunas de las tareas establecidas en la planificación inicial. A continuación, se muestra el Gantt diagrama que recoge los meses de marzo a junio. Los primeros meses del proyecto no se muestran porque no han sufrido ningún cambio respecto al primer diagrama 2.5.

Tal y como muestra el diagrama (8.1, 8.2, 8.3 y 8.4), las tareas de implementación y testeo se han realizado en paralelo ya que a medida que se creaba una *Feature* en la LPS, esta se testeaba hasta comprobar que funcionaba correctamente y empezar con la implementación de la siguiente característica.

[1] JavaScript Standard Style

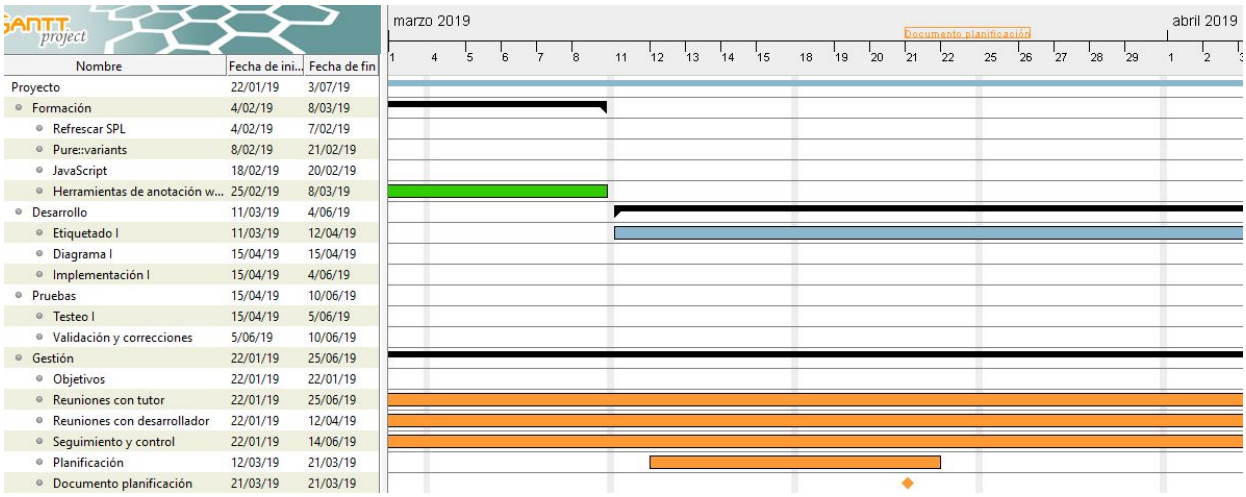


Figura 8.1: Parte del diagrama Gantt (1 de marzo - 1 de abril).

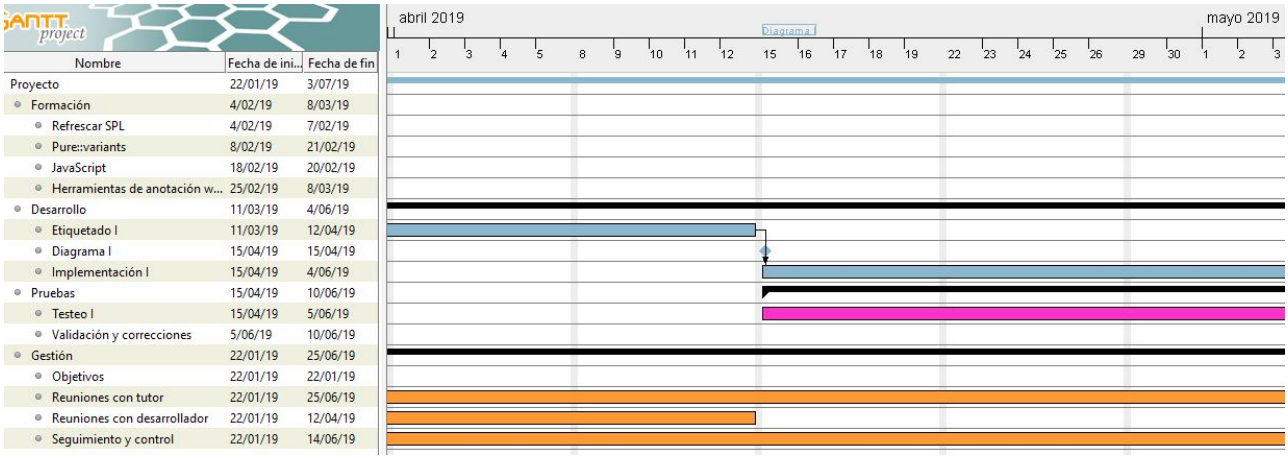
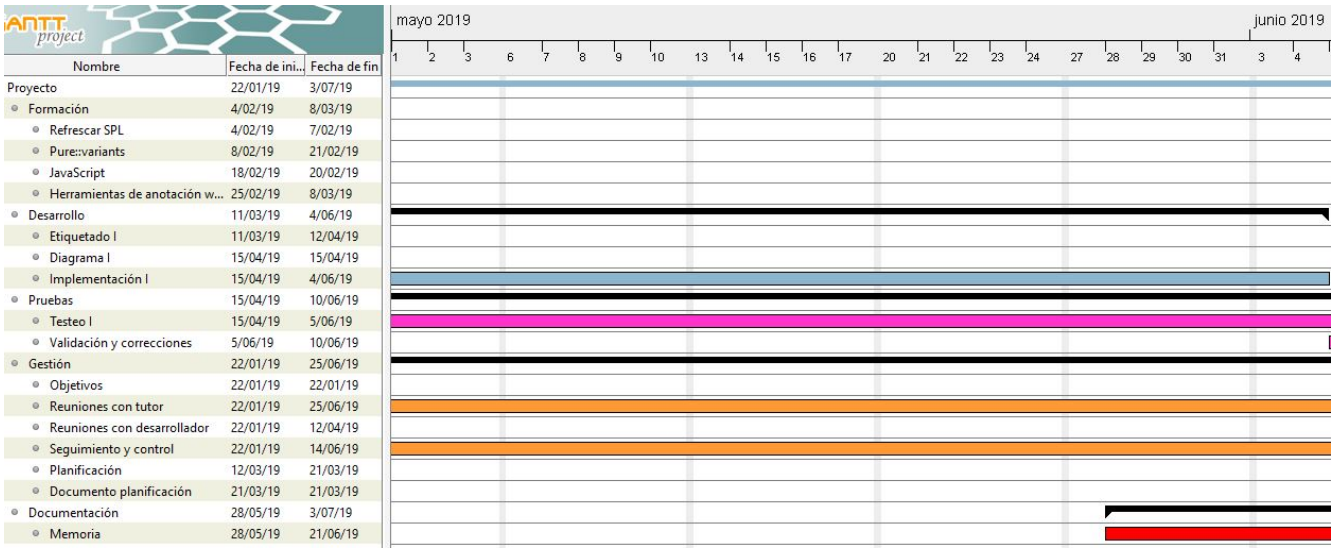
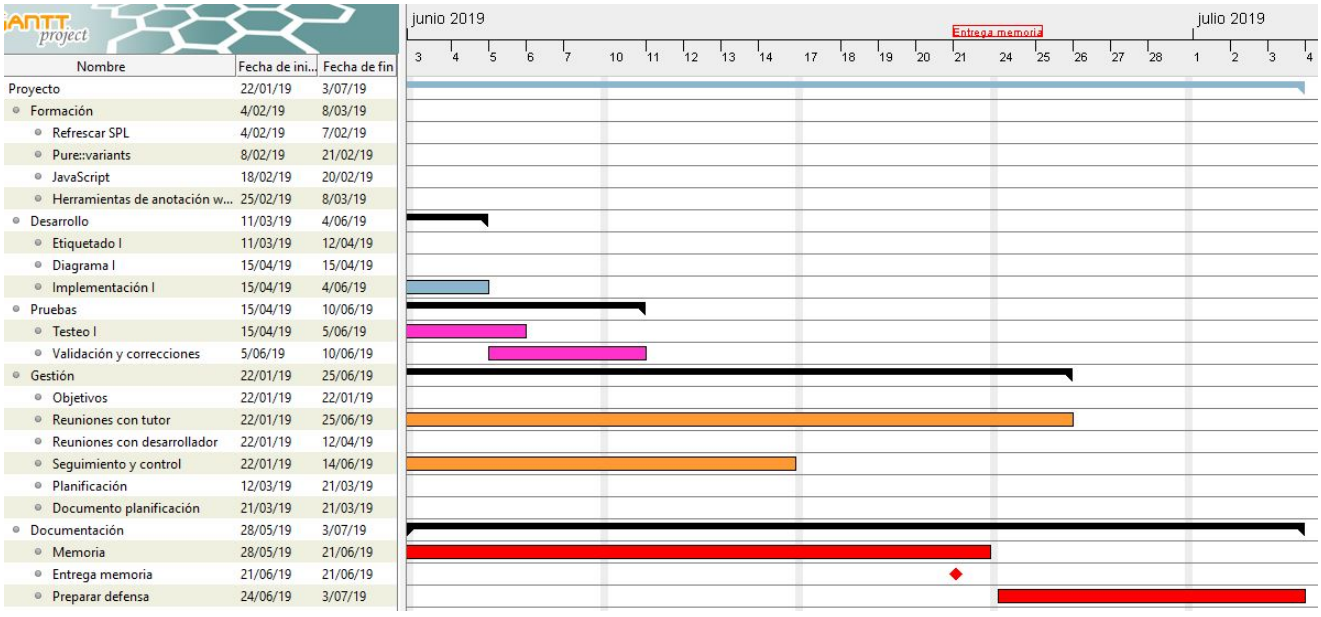


Figura 8.2: Parte del diagrama Gantt (1 de abril - 1 de mayo).



Nombre	Fecha de ini...	Fecha de fin
Proyecto	22/01/19	3/07/19
Formación	4/02/19	8/03/19
Refrescar SPL	4/02/19	7/02/19
Pure::variants	8/02/19	21/02/19
JavaScript	18/02/19	20/02/19
Herramientas de anotación w...	25/02/19	8/03/19
Desarrollo	11/03/19	4/06/19
Etiquetado I	11/03/19	12/04/19
Diagrama I	15/04/19	15/04/19
Implementación I	15/04/19	4/06/19
Pruebas	15/04/19	10/06/19
Testeo I	15/04/19	5/06/19
Validación y correcciones	5/06/19	10/06/19
Gestión	22/01/19	25/06/19
Objetivos	22/01/19	22/01/19
Reuniones con tutor	22/01/19	25/06/19
Reuniones con desarrollador	22/01/19	12/04/19
Seguimiento y control	22/01/19	14/06/19
Planificación	12/03/19	21/03/19
Documento planificación	21/03/19	21/03/19
Documentación	28/05/19	3/07/19
Memoria	28/05/19	21/06/19

Figura 8.3: Parte del diagrama Gantt (1 de mayo - 1 de junio).



Nombre	Fecha de ini...	Fecha de fin
Proyecto	22/01/19	3/07/19
Formación	4/02/19	8/03/19
Refrescar SPL	4/02/19	7/02/19
Pure::variants	8/02/19	21/02/19
JavaScript	18/02/19	20/02/19
Herramientas de anotación w...	25/02/19	8/03/19
Desarrollo	11/03/19	4/06/19
Etiquetado I	11/03/19	12/04/19
Diagrama I	15/04/19	15/04/19
Implementación I	15/04/19	4/06/19
Pruebas	15/04/19	10/06/19
Testeo I	15/04/19	5/06/19
Validación y correcciones	5/06/19	10/06/19
Gestión	22/01/19	25/06/19
Objetivos	22/01/19	22/01/19
Reuniones con tutor	22/01/19	25/06/19
Reuniones con desarrollador	22/01/19	12/04/19
Seguimiento y control	22/01/19	14/06/19
Planificación	12/03/19	21/03/19
Documento planificación	21/03/19	21/03/19
Documentación	28/05/19	3/07/19
Memoria	28/05/19	21/06/19
Entrega memoria	21/06/19	21/06/19
Preparar defensa	24/06/19	3/07/19

Figura 8.4: Parte del diagrama Gantt (1 de junio - 1 de julio).

9. CAPÍTULO

Conclusiones

En este último capítulo de la memoria se encuentran las conclusiones acerca del trabajo realizado para acometer el proyecto. Para ello, en primer lugar, se ha realizado una valoración de lo que ha sido el proyecto y de si ha cumplido las expectativas que estaban puestas en él. En segundo lugar, se ha realizado una valoración de los conocimientos adquiridos y sobre la experiencia que ha sido trabajar en este proyecto. Por último, se proponen las líneas por las que puede seguir creciendo la LPS desarrollada en un futuro y los puntos en los que puede mejorar.

9.1. Conclusión del proyecto

Tras finalizar el proyecto se han conseguido completar las expectativas que estaban propuestas desde un inicio. El proyecto se ha completado en un periodo de 5 meses y se ha conseguido cumplir con los objetivos dentro del plazo de entrega.

Aunque no se han podido implementar todas las *Features* que estaban contempladas en el *Feature Model*, la LPS desarrollada cuenta con 30 *Features* implementadas con las que se pueden generar una gran cantidad de productos diferentes. Además, la LPS diseñada está preparada para seguir siendo desarrollada y ya se han identificado nuevas *Features* que pueden ser interesantes de implementar.

La idea de añadir nuevas *Features* que aún no están implementadas es un ejemplo de la grandeza y crecimiento que puede llegar a lograr la Línea de Producto si se sigue traba-

jando sobre ella. La anotación Web es un recurso que ha sido concebido recientemente y tiene gran poder de crecimiento, ya que es un medio que se puede incorporar en muchas áreas con diferentes propósitos, lo cual hace que sea un dominio de gran escalabilidad. Existen y existirán diversas opciones de poder gestionar las anotaciones y de aplicar diferentes funcionalidades sobre ellas que harán crecer la Línea de Productos de la familia de anotadores Web creando nuevas variaciones de las características actuales a lo largo del tiempo.

Las *Features* que no se han implementado es por la falta de tiempo, el proyecto tiene establecido un máximo de horas e implementar todas las *Features* del proyecto se salía de las horas estipuladas. Además, el objetivo era crear un punto de partida para la LPS, por lo tanto, el objetivo se ha cumplido con creces.

Las mayores dificultades a la hora de realizar el proyecto se han encontrado en la fase de analizar el código. Entender un código de tres extensiones no ha sido tarea fácil, entre los tres productos suman una gran cantidad de archivos y los ficheros no contenían comentarios acerca del código que lo hiciesen más comprensible. No solo eso, si no que comprender un código en un lenguaje no muy familiar (JavaScript) ha dificultado la tarea. Para entender bien parte del código, ha habido que aprender conceptos nuevos como la gestión de eventos, llamadas asíncronas, promesas, callbacks o expresiones nuevas (lambda...). Además, aparte de entender el contenido del código de los ficheros también ha sido importante entender el funcionamiento de las extensiones, para ello ha sido imprescindible aprender cómo es su arquitectura y qué objetivo tiene cada fichero dentro de la aplicación.

Dentro de los productos de partida se ha encontrado mucho código que no se utilizaba y que se heredaba de los productos anteriores, esto provocaba bastante confusión al ir analizado el código porque no se sabía si el código era realmente necesario o no.

La ejecución de las extensiones también ha sido una tarea complicada. Tanto los productos de partida como los productos generados por la LPS se ejecutan desde el *framework* Node.js. Por lo tanto, al principio ha sido difícil aprender el proceso de preparar la extensión para ejecutarla en *Google Chrome* y ha llevado tiempo poder controlar bien el *framework*.

Se ha necesitado mucho tiempo para poder realizar las pruebas porque las extensiones tardan mucho en instalar todas las dependencias que necesitan y durante el proceso de generar productos ha habido muchos tiempos de espera.

Para entender bien el código de los productos de partida y realizar las refactorizaciones de

código que requerían grandes cambios ha sido fundamental la ayuda de Haritz Mediana, que siempre ha estado dispuesto a echar una mano.

9.2. Conocimientos adquiridos y experiencia personal

Ha sido la primera vez que he participado en un proyecto tan grande y tras estos 5 meses de trabajo la experiencia ha sido muy buena y ha superado las expectativas que tenía puestas sobre él. El proyecto ha merecido la pena porque siento que he aprendido mucho y he adquirido conocimientos que antes no tenía. A pesar de que a lo largo del proyecto he encontrado varias dificultades, estoy satisfecho porque he podido hacerles frente y he solicitado ayuda en los momentos que he requerido de ella.

Al principio existía el miedo de acomodarse en la realización del proyecto porque al tratarse de un trabajo que realiza uno por su propia cuenta y tener más asignaturas de por medio, temí que el proyecto se fuera retrasando, pero afortunadamente, tanto el proyecto como las asignaturas se han sacado adelante con éxito. En este tema la labor del tutor ha sido importante ya que ha tratado de que fuese al día con el trabajo.

El proyecto ha resultado beneficioso porque me ha aportado conocimientos de conceptos que no conocía, por ejemplo, al tratarse de una LPS de un dominio de anotaciones Web he podido aprender que son este tipo de anotaciones y para que pueden servir gracias a la ayuda de Haritz. También he podido ampliar conocimientos teóricos sobre el enfoque de Líneas de Productos y además aplicarlo a un caso de la vida real. Asimismo, en la realización del trabajo he aprendido a utilizar herramientas como *pure::variants*, una herramienta industrial, que al principio resulta un poco complicada de utilizar.

Al tratarse de una LPS de extensiones Web también he tenido la oportunidad de mejorar el manejo y adquirir mayores conocimientos de las tecnologías web como: JavaScript, CSS y HTML. Además, he podido aprender cuál es la arquitectura de una extensión Web y que elementos la componen.

Al ser un proyecto grande siento que he ganado experiencia profesional en cuanto a planificarme el proyecto, realizar reuniones, rellenar actas e ir dando los pasos necesarios hasta alcanzar el objetivo. Siempre con la ayuda de mi tutor Óscar y Haritz, con los que he formado equipo para realizar correctamente el proyecto.

Siento que las reuniones con Haritz para realizar las implementaciones han sido muy beneficiosas porque aparte de enseñarme y ayudarme a realizar las implementaciones

necesarias también me han aportado muchos más conocimientos con temas que cubren el proyecto o que no eran parte de él.

9.3. Mejoras y líneas futuras

El objetivo de este proyecto era que al finalizar el proyecto quedase desarrollada una LPS que da soporte a los tres productos actuales y además ayudase a que en un futuro de cabida a nuevas funcionalidades y nuevos cambios necesarios. Tal y como ha quedado la LPS se consigue cubrir la mayoría de las *Features* así que una de las mejoras sería terminar de implementar las pocas *Features* restantes que han quedado por hacer.

Otra de las posibles mejoras es poder crear más tipos de scopes para las diferentes visualizaciones que existen para que las *Features* que se agrupan en el grupo de Scope tengan más sentido.

Por otro lado, también sería interesante extraer el código de las *Features* obligatorias de las aplicaciones para que en un futuro estas puedan ser variables. Por ejemplo, durante el proyecto se ha conseguido que *Hypothesis* sea una *Feature* más gracias a que se ha implementado la *Feature* Local. Siguiendo esta línea, también podría ser interesante crear más tipos de almacenamiento externo para poder elegir otro tipo almacén de anotaciones que no sea Hypothes.is.

Siguiendo con el hilo de implementar más *Features* en un futuro sería interesante implementar características como la de poder importar/exportar grupos, compartir los grupos y etcétera. Al crearse nuevas funcionalidades para las aplicaciones Haritz tendrá la opción de implementar las *Features* sobre la LPS en vez de tener que crear nuevos productos como hasta ahora.

Anexos

A. ANEXO

Actas de reuniones

En este anexo se encuentran las actas de las reuniones que se han realizado durante el proyecto. Estas actas se han escrito tras finalizar las reuniones con el objetivo de ir guardando los puntos más importantes que se han hablado durante la reunión.

Las actas pertenecen a las reuniones que se han realizado con el tutor para hacer el seguimiento del proyecto y las reuniones en las que ha participado Haritz con el objetivo de presentar las herramientas o resolver dudas acerca de los productos de partida.

En las actas se recogen los asistentes a la reunión, los temas a tratar, el resumen de lo que se ha hablado en la reunión y los deberes asignados para la siguiente reunión.

La primera acta pertenece a la primera reunión que se realizó para presentar el proyecto y las herramientas que se debían utilizar para llevar a cabo el proyecto.

No se han realizado actas de las reuniones realizadas con Haritz que tenían como objetivo realizar la implementación de refactorizaciones y cambios en el proyecto.

A.1. Acta 1

Presentación del proyecto

Fecha: 22/01/2019 Hora: 15:15 Duración: 2h Lugar: Seminario 3.2

Asistentes:

- Xabier Garmendia
- Mikel Cantero
- Haritz Medina
- Raúl Medeiros
- Arantza Irastorza

Temas a tratar:

- Conocer los objetivos del proyecto.
- Introducción en la herramienta pure::variants mediante la ayuda de Raúl.
- Introducción en el tema de las anotaciones Web de la mano de Haritz Medina.

Resumen de los temas tratados:

Se ha hablado sobre el alcance del proyecto, se tratará de realizar una Línea de Productos aplicada a la familia de productos de extensiones de anotación Web creadas por Haritz Medina.

Raúl ha dado a conocer a Xabi y Mikel la herramienta pure::variants, que será necesaria a la hora de hacer el proyecto. Para ello, se han realizado varios ejemplos para comprender su funcionamiento.

Haritz ha presentado sus productos a los alumnos. Ha explicado sus funcionalidades mediante pruebas y ha mostrado cómo instalar cada extensión en el navegador para poder usarlas. También ha facilitado el código de cada producto, que se encuentran alojados en la plataforma GitHub.

Deberes asignados:

- **Xabier y Mikel**

  -Tratarán de realizar pruebas con pure::variants para conocer mejor la herramienta.

  -Tratarán de utilizar los productos de anotación Web para familiarizarse con ellas.

- **Xabier, Mikel y Haritz**

  -Se ha acordado fijar otra reunión para finalizar de conocer los productos de anotación Web, ya que no ha dado tiempo para poder acabar de ver todo.

A.2. Acta 2

2. Reunión para terminar la presentación del proyecto.

Fecha: 18/02/2019 Hora: 10:30 Duración: 3h Lugar: Seminario 3.2

Asistentes:

- Xabier Garmendia
- Mikel Cantero
- Haritz Medina

Temas a tratar:

- Terminar de conocer y probar la herramienta *Mark&Go*.
- Tratar las dudas surgidas a la hora de entender el código o el uso de los productos.
- Conocer la *plug-in* llamado *but4reuse* para facilitar la comparación entre los códigos de los productos.
- Conocer cómo instalar una extensión en el navegador.

Resumen de los temas tratados:

Haritz ha terminado de explicar la herramienta *Mark&Go*, a parte también se ha realizado un repaso del uso de los otros dos productos para refrescar la memoria y zanjar dudas sobre su uso.

Mikel y Xabi han aprendido cómo obtener el código de los productos, como compilarlo y como instalarlo como una extensión del navegador con la colaboración de Haritz.

Finalmente, Haritz ha dado a conocer el *plug-in* de Eclipse llamado *but4reuse* mediante unos ejemplos y ha recomendado otra herramienta interesante como DiffMerge.

Deberes asignados:

- **Xabier y Mikel**

  -Una vez conocidos los productos, en la reunión con el tutor establecer cuál será el próximo paso.

  -Descargar los códigos de *Highlight&Go* y *Review&Go*

- **Haritz**

  -Terminará de completar el código de la herramienta *Mark&Go* y avisará cuando esté terminado para poder guardarlo y analizarlo.

A.3. Acta 3

1. Reunión con Óscar.

Fecha: 18/02/2019 Hora: 13:00 Duración: 30 min Lugar: Despacho de Óscar

Asistentes:

- Xabier Garmendia
- Óscar Díaz

Temas a tratar:

- Una vez conocidas las herramientas, hablar sobre cuál será el siguiente paso a realizar para empezar con la realización del proyecto.

Resumen de los temas tratados:

Se han establecido los próximos pasos a seguir durante la semana:

Paso 1: Terminar de familiarizarse con la funcionalidad de los 3 productos: *Highlight&Go*, *Mark&Go* y *Review&Go*.

Paso 2: Entender el código de dos productos: *Highlight&Go* y *Review&Go*İmportante: repasar JavaScript. Entender bien qué hace el código.

Paso 3: Obtener las diferencias de código entre estos dos productos. Para ello utilizar DiffMerge. Los productos tienen alrededor de 40 ficheros. Por lo tanto, se aplicará Diff a los 40 productos. Generar un fichero PDF por cada Diff.

Paso 4: Identificación de características. Para soportar una característica, puede ser necesario tocar varios ficheros. A las líneas de código de un fichero que soportan una característica, se le denomina "punto de variación". En este paso, hay que identificar qué puntos de variación de distintas características se corresponden con la misma característica. Para ello, a cada una, se le asignará un color. Utilizando *Review&Go*, se irá subrayando los distintos ficheros, y en concreto las líneas de código, con el color que creamos que sirve para soportar cada una de las características.

Deberes asignados:

- **Xabier**

  -Entender el código de dos productos: *Highlight&Go* y *Review&Go*.

  -Sacar características de los dos productos y tratar de realizar un primer diagrama de las características, conocido como *Feature Model*.

  -Identificar las características con la herramienta de *Review&Go*, para así familiarizarse más con ella.

  -Una vez terminados los pasos, establecer una nueva reunión con el tutor.

A.4. Acta 4

1. Reunión con Haritz para resolver dudas.

Fecha: 25/02/2019 Hora: 11:00 Duración: 3 h Lugar: Seminario 3.2

Asistentes:

- Xabier Garmendia
- Haritz Medina

Temas a tratar:

- Resolver dudas sobre *Highlight&Go* y *Review&Go* al no poder utilizarlas bien.

Resumen de los temas tratados:

Se ha conseguido ejecutar la extensión de *Review&Go*, pero en *Highlight&Go* se ha localizado un error por un cambio realizado en una API externa que no permite el correcto funcionamiento.

Deberes asignados:

- **Xabier**

  -Una vez haber conseguido ejecutar la extensión *Review&Go*, comenzar a utilizarla para descubrir funcionalidades y empezar a analizar el código de *Review&Go* y *Highlight&Go*.

- **Haritz**

  -Tratará de arreglar el problema de *Highlight&Go*.

A.5. Acta 5

2. Reunión con Óscar.

Fecha: 12/03/2019 Hora: 15:00 Duración: 30min Lugar: Despacho de Óscar

Asistentes:

- Xabier Garmendia
- Óscar Díaz

Temas a tratar:

- Comentar el estado del proyecto y conocer la herramienta GanttProject

Resumen de los temas tratados:

Se ha comentado cómo se encuentra el estado del análisis del código y se ha quedado para terminarlo en los próximos días. Por otro lado, para poder observar la planificación inicial Óscar ha recomendado la herramienta GanttProject.

Deberes asignados:

- **Xabier**

  -Continuar con las tareas pendientes y enviar a Óscar la documentación de la planificación.

- **Óscar**

  -Cuando Xabi le mande los documentos de planificación se encargará de revisarlos.

A.6. Acta 6

2. Reunión con Haritz para resolver dudas.

Fecha: 28/03/2019 Hora: 15:30 Duración: 3h Lugar: Seminario 3.2

Asistentes:

- Xabier Garmendia
- Haritz Medina

Temas a tratar:

- Dudas sobre el código de los productos y entender el funcionamiento de algunos de los ficheros más complicados del proyecto.

Resumen de los temas tratados:

Haritz ha explicado cómo están formados los ficheros principales que son muy complejos para entender. Por otro lado, se ha valorado cambiar la manera de analizar los productos, en vez de analizar solamente dos de los productos puede que realizar el análisis de los tres a la vez sea mejor para tener ya el prototipo de *Feature Model* cuanto antes.

Deberes asignados:

- **Xabier**

  -Xabi seguirá con el proceso de analizar el código para tener pronto el prototipo del *Feature Model*.

  -Comenzará a analizar el código de *Mark&Go* junto a los demás.

- **Xabier y Haritz**

  -En la reunión de mañana con Óscar le comentarán los cambios de planificación que se han planteado.

A.7. Acta 7

Reunión de seguimiento del proyecto.

Fecha: 29/03/2019 Hora: 15:00 Duración: 30 min Lugar: Cafetería de la Facultad.

Asistentes:

- Xabier Garmendia
- Óscar Díaz
- Haritz Medina

Temas a tratar:

- Comentar en qué punto se encuentra el proyecto.

Resumen de los temas tratados:

Se ha decidido cambiar la manera de efectuar el desarrollo. Ahora se hará el análisis y la implementación de los tres productos a la vez.

Se ha comentado empezar a escribir parte de la memoria para que en la recta final no se tenga que hacer todo de golpe.

Deberes asignados:

- **Xabier**

  -Tratará de terminar el análisis cuanto antes para tener el prototipo del *Feature Model* entero de la LPS.

- **Haritz**

  -Enviará documentación a Xabi que le pueda ser valiosa para empezar a realizar el capítulo donde se presentan los productos.

A.8. Acta 8

3. Reunión con Haritz para resolver dudas.

Fecha: 04/04/2019 Hora: 11:00 Duración: 3h Lugar: Seminario 3.2

Asistentes:

- Xabier Garmendia
- Haritz Medina

Temas a tratar:

- Resolver las dudas surgidas a Xabi sobre los ficheros que corresponden a la configuración de Moodle en *Mark&Go*.

Resumen de los temas tratados:

Haritz ha explicado a Xabi las funciones de los ficheros que corresponden a Moodle. Además, se han comentado más dudas que han surgido de los otros productos.

Deberes asignados:

- **Xabier**

  -Una vez entendida la parte de *Mark&Go* tratará de terminar el diseño del *Feature Model* para cuando termine la semana de trabajos en sus asignaturas.

A.9. Acta 9

4. Reunión con Haritz para resolver dudas.

Fecha: 15/04/2019 Hora: 10:30 Duración: 3h 30min Lugar: Seminario 3.2

Asistentes:

- Xabier Garmendia
- Haritz Medina

Temas a tratar:

- Resolver las últimas dudas sobre algunos ficheros y mejorar el prototipo del *Feature Model* realizado por Xabi para presentárselo a Óscar en la reunión de la tarde.

Resumen de los temas tratados:

Se ha terminado de realizar el primer prototipo del *Feature Model*.

Deberes asignados:

- **Xabier y Haritz**

  -En la reunión de la tarde presentarán a Óscar el prototipo acordado entre los dos.

A.10. Acta 10

1. Reunión para definir el *Feature Model*.

Fecha: 15/04/2019 Hora: 15:00 Duración: 1h 30min Lugar: Despacho de Óscar

Asistentes:

- Xabier Garmendia
- Óscar Díaz
- Haritz Medina

Temas a tratar:

- Se ha comentado el *Feature Model* planteado por Haritz y Xabi.

Resumen de los temas tratados:

Se han realizado varios cambios en el *Feature Model* y se ha decidido quedar al día siguiente para terminar de definir el esquema definitivo.

Deberes asignados:

- **Xabier, Haritz y Óscar**

  Pensar ideas para terminar de completar el *Feature Model*.

A.11. Acta 11

2. Reunión para definir el *Feature Model*.

Fecha: 17/04/2019 Hora: 15:00 Duración: 1h 30min Lugar: Despacho de Óscar.

Asistentes:

- Xabier Garmendia
- Óscar Díaz
- Haritz Medina

Temas a tratar:

- Terminar de definir el *Feature Model*.

Resumen de los temas tratados:

Se ha terminado de elaborar el *Feature Model* entre los asistentes a la reunión.

Deberes asignados:

- **Xabier**

  Documentar el *Feature Model* y empezar la implementación a la vuelta de la Semana Santa.

B. ANEXO

Manual de instalación de los productos generados

En este último anexo se encuentran las instrucciones necesarias para poder instalar las extensiones web generadas por la LPS en el navegador de *Chrome* para poder realizar anotaciones.

En primer lugar, es necesario tener instalado en el ordenador las herramientas *npm* y *Gulp*.

npm es un sistema que se encarga de gestionar las dependencias para Node.js que se instala automáticamente al descargarse el entorno de Node.js. La ejecución de *npm* se realiza desde la línea de comandos y sirve para instalar las dependencias de las aplicaciones.

Por otro lado, Gulp es una herramienta que ayuda a automatizar tareas comunes en el desarrollo de una aplicación. Por ejemplo: compilar dependencias, preprocesado de hojas de estilo, empaquetado de extensiones, testing, etcétera...

Para instalarlo es necesario instalarlo tanto a nivel global como a nivel de proyecto. Para instalarlo de manera global se debe ejecutar el siguiente comando:

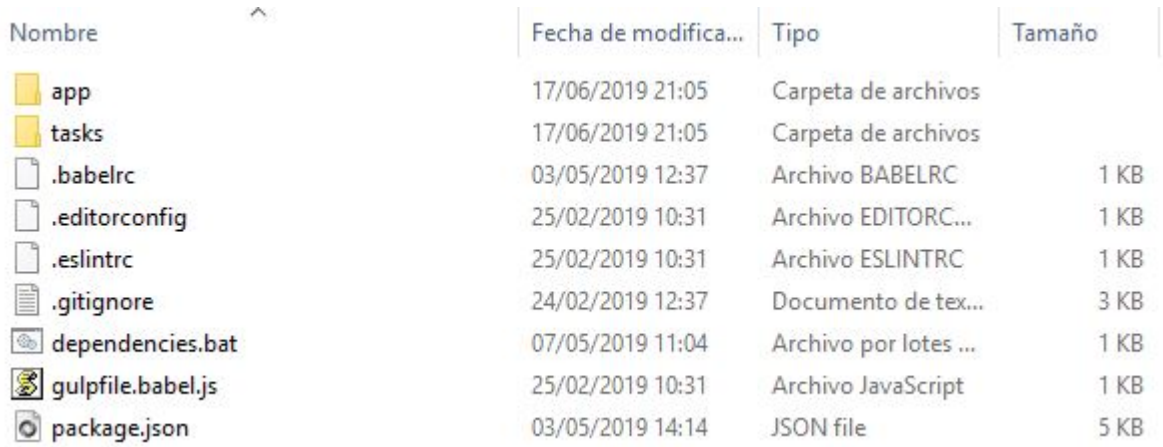
```
npm install -g gulp
```

Por otro lado, para instalar Gulp a nivel de proyecto se utiliza el siguiente comando:

```
npm install --save-dev gulp
```

Una vez que se dispone de las herramientas ya se puede proceder a preparar la extensión para poder utilizarla.

Para ello, en primer lugar, nos debemos colocar en la carpeta del producto que se ha generado desde la LPS y abrir la línea de comandos.



Nombre	Fecha de modifica...	Tipo	Tamaño
app	17/06/2019 21:05	Carpeta de archivos	
tasks	17/06/2019 21:05	Carpeta de archivos	
.babelrc	03/05/2019 12:37	Archivo BABELRC	1 KB
.editorconfig	25/02/2019 10:31	Archivo EDITORC...	1 KB
.eslintrc	25/02/2019 10:31	Archivo ESLINTRC	1 KB
.gitignore	24/02/2019 12:37	Documento de tex...	3 KB
dependencies.bat	07/05/2019 11:04	Archivo por lotes ...	1 KB
gulpfile.babel.js	25/02/2019 10:31	Archivo JavaScript	1 KB
package.json	03/05/2019 14:14	JSON file	5 KB

Figura B.1: Carpeta generada por la LPS que pertenece a un producto.

El primer comando que se debe ejecutar es el siguiente:

```
1 npm install
```

Este comando se encargará de descargar todas las dependencias que se encuentran en el fichero package.json ya que reúne todas las dependencias que se pueden encontrar en los productos.

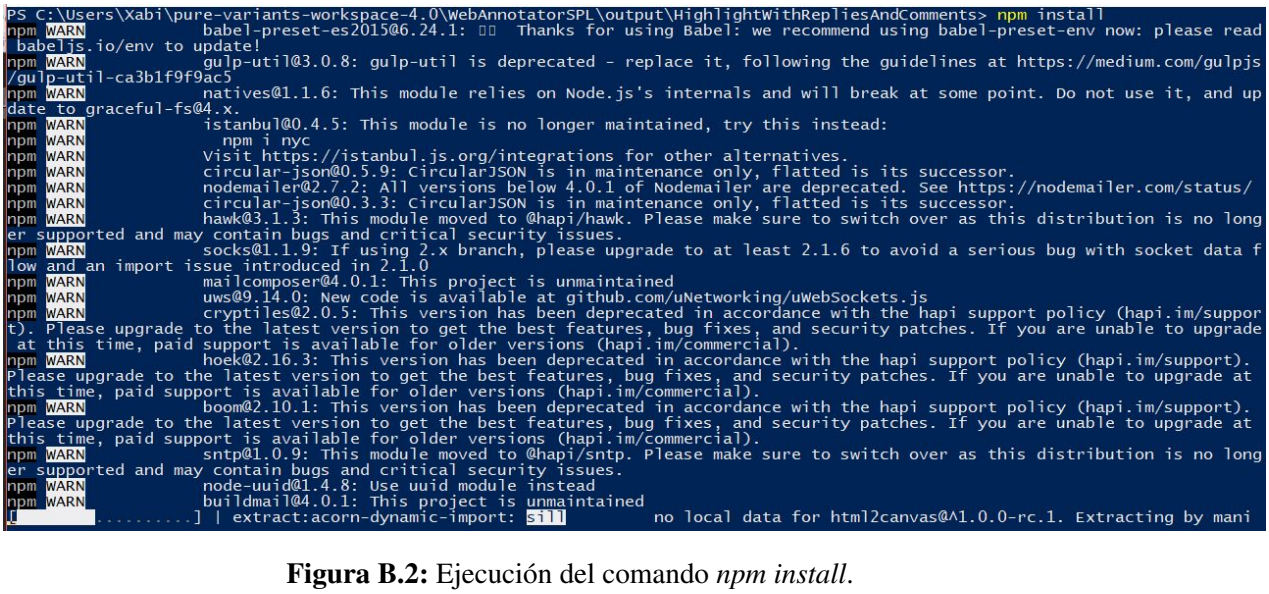
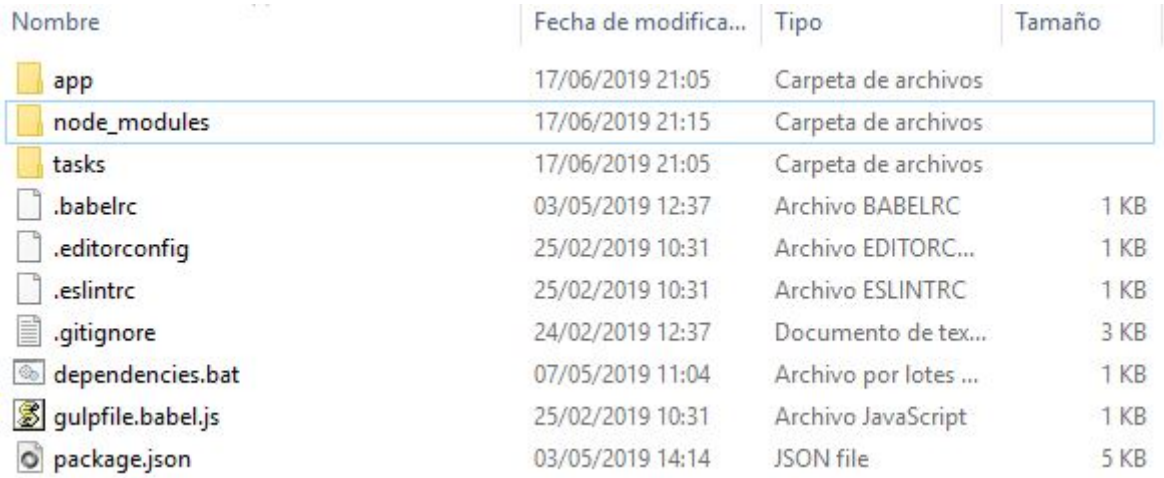


Figura B.2: Ejecución del comando *npm install*.

Cuando el proceso termina su ejecución se habrá creado una carpeta nueva con el nombre *node_modules*, que reúne las carpetas que corresponden a cada dependencia.



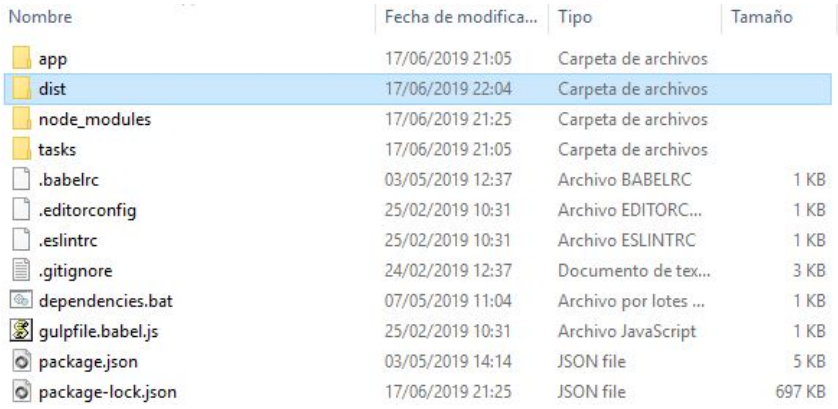
Nombre	Fecha de modifica...	Tipo	Tamaño
app	17/06/2019 21:05	Carpeta de archivos	
node_modules	17/06/2019 21:15	Carpeta de archivos	
tasks	17/06/2019 21:05	Carpeta de archivos	
.babelrc	03/05/2019 12:37	Archivo BABELRC	1 KB
.editorconfig	25/02/2019 10:31	Archivo EDITORC...	1 KB
.eslintrc	25/02/2019 10:31	Archivo ESLINTRC	1 KB
.gitignore	24/02/2019 12:37	Documento de tex...	3 KB
dependencies.bat	07/05/2019 11:04	Archivo por lotes ...	1 KB
gulpfile.babel.js	25/02/2019 10:31	Archivo JavaScript	1 KB
package.json	03/05/2019 14:14	JSON file	5 KB

Figura B.3: El estado de la carpeta tras ejecutar el comando *npm install*.

El siguiente paso es crear una carpeta comprimida para instalarla en el navegador. Para conseguir esta carpeta se debe ejecutar la siguiente sentencia en la línea de comandos:

```
1 gulp default
```

La tarea "gulp default" se encargará de compilar la extensión comprimiendo los ficheros del proyecto en una nueva carpeta y revisando los errores que se encuentran en el código. Una vez termina la ejecución si no ha ocurrido ningún error, se habrá creado una nueva carpeta que contiene los ficheros de ejecución que hay que instalar en el navegador.



Nombre	Fecha de modifica...	Tipo	Tamaño
app	17/06/2019 21:05	Carpeta de archivos	
dist	17/06/2019 22:04	Carpeta de archivos	
node_modules	17/06/2019 21:25	Carpeta de archivos	
tasks	17/06/2019 21:05	Carpeta de archivos	
.babelrc	03/05/2019 12:37	Archivo BABELRC	1 KB
.editorconfig	25/02/2019 10:31	Archivo EDITORC...	1 KB
.eslintrc	25/02/2019 10:31	Archivo ESLINTRC	1 KB
.gitignore	24/02/2019 12:37	Documento de tex...	3 KB
dependencies.bat	07/05/2019 11:04	Archivo por lotes ...	1 KB
gulpfile.babel.js	25/02/2019 10:31	Archivo JavaScript	1 KB
package.json	03/05/2019 14:14	JSON file	5 KB
package-lock.json	17/06/2019 21:25	JSON file	697 KB

Figura B.4: El estado de la carpeta tras ejecutar el comando *gulp default*.

En la figura B.4 se muestra cómo queda la carpeta del proyecto tras la ejecución del comando anterior. En ella, se puede observar que aparece una carpeta nueva llamada *dist*. Dentro de la carpeta *dist* se encuentran los ficheros que hay que instalar en el navegador.

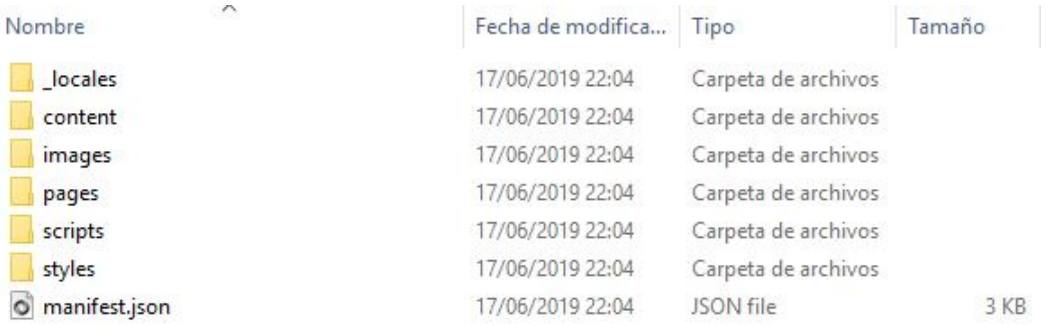


Figura B.5: Carpetas creadas tras la ejecución de *gulp default* en la que se encuentran los ficheros de ejecución.

Para instalar la carpeta en el navegador se debe abrir la pestaña de extensiones en la que se encuentran todas las extensiones instaladas.

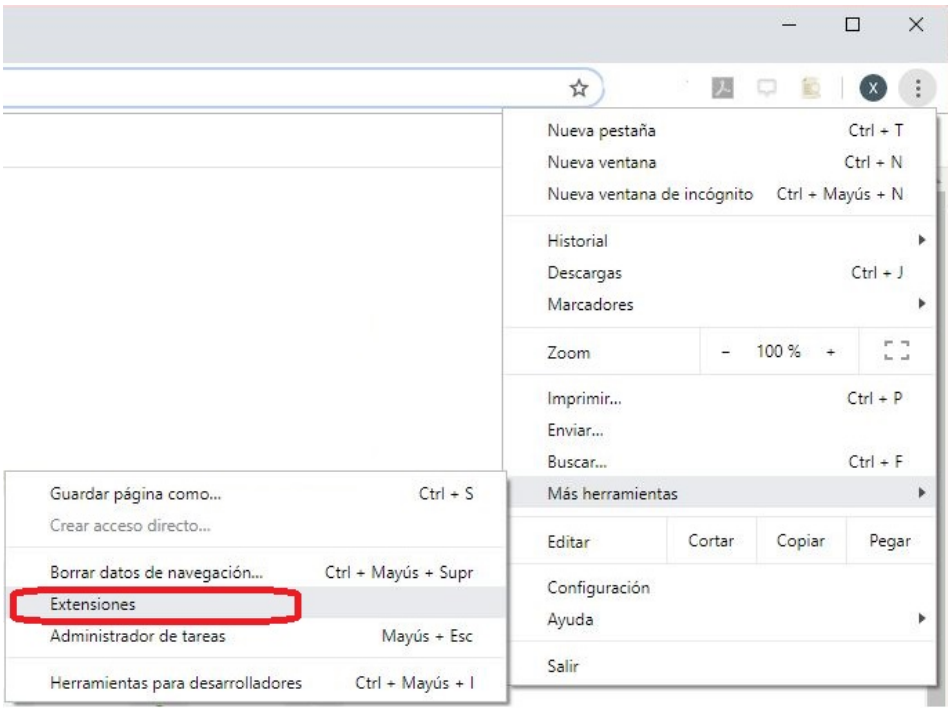


Figura B.6: Pestaña que se debe abrir para que se muestren las extensiones.

En la figura B.6 se muestra cual es la opción que se debe elegir para abrir la ventana

de gestionar extensiones. En la nueva ventana que se abre, se debe elegir la opción de "Cargar descomprimida" y seleccionar la carpeta que contiene los ficheros de ejecución.

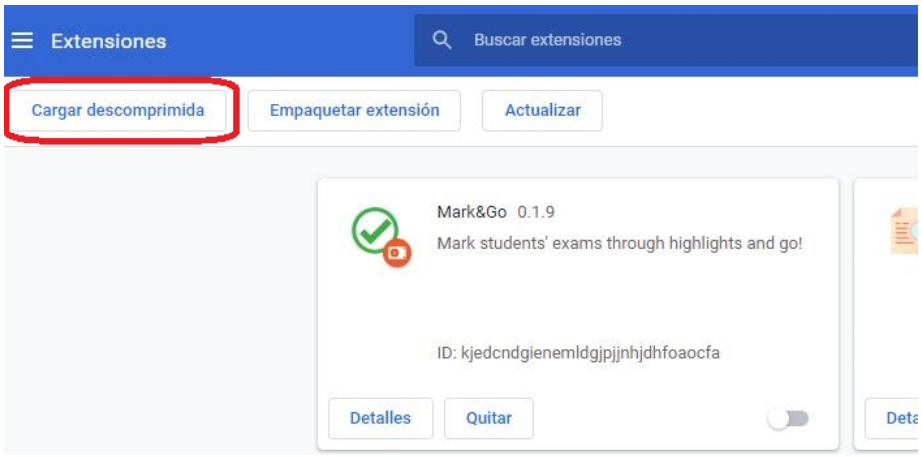


Figura B.7: Botón que se debe pulsar para instalar la carpeta de la extensión.

Una vez seguidos los pasos, la extensión estará instalada en el navegador y ya se puede empezar a utilizar.

Bibliografía

[Beuche and Dalgarno, 2019] Beuche, D. and Dalgarno, M. (2019). *Software Product Line Engineering with Feature Models*. https://www.pure-systems.com/fileadmin/downloads/pure-variants/tutorials/SPLWithFeatureModelling.pdf.

[Díaz et al., 2019a] Díaz, O., Contell, J. P., and Medina, H. (2019a). Performant Peer Review for Design Science Manuscripts: A Pilot Study on Dedicated Highlighters. *DESRIST 2019. Lecture Notes in Computer Science, vol 11491.*

[Díaz et al., 2019b] Díaz, O., Medina, H., and Anfurrutia, F. I. (2019b). Coding-Data Portability in Systematic Literature Reviews: a W3C's Open Annotation Approach. *EASE '19 Proceedings of the Evaluation and Assessment on Software Engineering*.

[Diaz and Trujillo, 2010] Diaz, O. and Trujillo, S. (2010). Líneas de producto software. In Piattini, M. G. and Garzás, J., editors, *Fábricas de Software: Experiencias, técnologias y organización*, chapter 3, pages 121–141. Ra-Ma Editorial, S.A, Madrid, 2 edition.

[do Nascimento et al., 2019] do Nascimento, L. M., de Almeida1, E. S., and de Lemos Meira, S. R. (2019). Core Assets Development in Software Product Lines - Towards a Practical Approach for the Mobile Game Domain. *SBCARS 2009*.

[Google, 2019] Google (2019). *Getting Started Tutorial*. https://developer.chrome.com/extensions/getstarted.

[Hypothes.is, 2019a] Hypothes.is (2019a). *5 Million Annotations*. https://web.hypothes.is/blog/5-million-annotations/.

[Hypothes.is, 2019b] Hypothes.is (2019b). *To enable a conversation over the world's knowledge*. https://web.hypothes.is/about/.

[Mark&Go, 2019] Mark&Go (2019). Scalability & Quality Tradeoffs in Student Feedback: WebAnnotations to the rescue.

[Ouali et al., 2019] Ouali, S., Kraiem, N., and Ghezala, H. B. (2019). Framework for Evolving Software Product Line. *ResearchGate*.

[Pérez, 2014a] Pérez, M. M. (2014a). *Bibliografía en LATEX*. http://logistica.fime.uanl.mx/miguel/docs/BibTeX.pdf.

[Pérez, 2014b] Pérez, M. M. (2014b). *Bibliografía en LATEX*. http://logistica.fime.uanl.mx/miguel/docs/BibTeX.pdf.

[pure systems, 2019a] pure systems (2019a). *pure::variants Connector for Source Code Management Manual*. http://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-conn-sourcecode-manual.pdf.

[pure systems, 2019b] pure systems (2019b). *pure::variants User's Guide*. https://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.pdf.

[SourceGear, 2017] SourceGear (2017). *DiffMerge*. https://sourcegear.com/diffmerge/.

[W3C, 2017a] W3C (2017a). *Web annotation Data Model*. https://www.w3.org/TR/annotation-model/.

[W3C, 2017b] W3C (2017b). *Web annotation Protocol*. https://www.w3.org/TR/annotation-protocol/.

[W3C, 2017c] W3C (2017c). *Web annotation Vocabulary*. https://www.w3.org/TR/annotation-vocab/.

[Wikipedia, 2019a] Wikipedia (2019a). *Hypothes.is*. https://es.wikipedia.org/wiki/Hypothes.is.

[Wikipedia, 2019b] Wikipedia (2019b). *Software product line*. https://en.wikipedia.org/wiki/Software_product_line.

[Wikipedia, 2019c] Wikipedia (2019c). *Web annotation*. https://en.wikipedia.org/wiki/Web_annotation.