UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Aplicación de soporte a personas con dificultades de visión

Presentado por Bryan Reinoso Cevallos
en Universidad de Burgos — Julio de 2014
Tutores: José Francisco Díez Pastor,
Dr. César I. García Osorio

Resumen

Descriptores

Abstract

Keywords

Índice general

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

El objetivo principal del proyecto consiste en la creación de una aplicación de soporte para personas con dificultades de visión.

La aplicación de soporte para los usuarios será desarrollada en Android y constará de una interfaz y funcionamiento orientada al uso por personas con dificultades de visión, además de guiará al usuario a través de la aplicación con un asistente de voz que irá diciendo al usuario qué debe hacer y en qué punto de la aplicación se encuentra.

La aplicación tendrá como objetivo inicial mandar una foto tomada por el usuario a un servidor que la procesará y extraerá una predicción que le dirá al usuario lo que se puede observar en la imagen que él ha elegido. La predicción será una frase que describa la imagen y está será leída por la aplicación.

Por otra parte en el lado del servidor usaremos algoritmos de machine learning para el procesado de la imagen y la extracción de la predicción. El servidor recibirá una petición de tipo POST y cargará la imagen, la cuál será procesada y en respuesta devolverá una frase en español que describa la imagen.

El principal objetivo del proyecto es conseguir crear un prototipo de esta aplicación que tenga un correcto funcionamiento y que sirva como base para desarrollar sobre él una aplicación más optimizada y de aún mejor funcionamiento.

2. OBJETIVOS DEL PROYECTO

2.1 Objetivos funcionales

El principal objetivo del proyecto es tener un prototipo de la aplicación que funcione de manera adecuada,para esto tenemos que tener en cuenta tanto los objetivos del lado del servidor como del lado del cliente.

2.1.1 Objetivos funcionales en el cliente Android

El principal objetivo que podemos encontrarnos en el lado del cliente es el disponer de un prototipo de aplicación Android con la que el usuario pueda mandar de manera intuitiva una imagen al servidor y recibir de él la predicción esperada. Para que esto funcione de manera adecuada se han propuesto una serie de requisitos que deben cumplirse:

- Para poder enviar la imagen al servidor y recibir de él la predicción, se debe establecer una conexión con el desde la aplicación Android y hacer un correcta petición de tipo POST. Para ello usamos la librería de Apache con el módulo MultipartEntityBuilder, que nos permite establecer de manera bastante sencilla la conexión y mandar la petición POST. Además por especficaciones de Android esto deberá realizarse en un hilo separado del hilo principal.
- Para que la persona que lo utilice no tenga por qué saber la estructura de la interfaz gráfica de la aplicación para poder usarla sin problemas, nos apoyaremos sobre los eventos ontouch de Android y evitaremos lo máximo posible que el usuario tenga que saber cómo es la interfaz.
- Para ofrecer una guía fácil y útil para el usuario a lo largo de su experiencia usando la aplicación, usaremos la librería Text2Speech para ir guiando al usuario con instrucciones en voz alta explicándole lo que debe hacer en cada momento. Finalmente le devolveremos la predicción y se le dirá cuáles son sus opciones.
- Para poder tomar la imagen se requerirá de un dispositivo con cámara de fotos y se tendrá que establecer los permisos sobre la aplicación para el uso de la misma.

2.1.2 Objetivos funcionales en el servidor Flask

En el lado del servidor tenemos un objetivo claro y es el de recibir una imagen a través de una petición POST desde un cliente, procesarla y, finalmente, devolver la predicción en español. Para conseguir todo esto tenemos una serie de requisitos y objetivos que tenemos que cumplir:

- Para poder recibir las peticiones y mandar respuestas tenemos que tener un servidor con la capacidad de gestionar este tipo de operaciones, para ello se ha decidido usar Flask para programar el servidor. Con Flask se creará un servidor que gestione la petición post sobre un URL concreto .

- Para poder procesar la imagen se necesita una red neuronal a través de la cual pasar la imagen y extraer la predicción. Para ello se ha elegido un proyecto desarrollado por Karpathy y su proyecto Neuraltalk. Para ello se instalará el proyecto en la máquina servidora y todas las dependencias del mismo. Además se hará todas las modificaciones necesarias para su correcto funcionamiento sobre nuestra aplicación.
- Una vez la predicción haya sido extraída esta viene en inglés, así que el servidor se deberá encargar de traducirla al español para enviarla al cliente. Para ello se usa una conexión con el traductor de Google.

3. CONCEPTOS TEÓRICOS

En este apartado se profundizará en los conceptos teóricos con los que se ha trabajado a lo largo de todo el proyecto.

3.1 Conceptos teóricos en el lado del Servidor

En este apartado se pretende explicar todos los conceptos teóricos que se utilizan en el lado del servidor, tanto directamente usados por el alumno como los que se usan en proyectos o librerías de apoyo que se usan en el proyecto.

3.1.1 Machine Learning

El *Machine Learing* o también conocido como aprendizaje computacional, entre otros nombres, es una rama de la inteligencia artificial que pretende conseguir el objetivo de que los computadores puedan aprender de manera automática. De forma más general se podría decir que en el *machine learning* se pretende crear programas informáticos capaces de generalizar comportamientos a partir de una información no estructurara que suele estar suministrada en forma de ejemplos.

El *machine learning* tiene muchísimas aplicaciones y es ahora mismo un campo de la inteligencia artificial que se encuentra en estudio y del que se suceden bastantes proyectos.

■ Tipos de algoritmos

Podemos dividir los algoritmos en tres tipos, aunque puede haber más pero los más importantes o con los que se puede clasificar a todos los algoritmos o programas que nos encontremos:

- **Aprendizaje Supervisado:** Este algoritmo es modelado con una serie de ejemplos que son la entrada del sistema, pero también tienen una etiqueta de salida para cada ejemplo. El sistema lo que intenta es predecir la salida que se corresponde con cada ejemplo a través de su entrada, en función del error o acierto a la hora de predecir, el algoritmo va modificando sus valores para generar cada vez una predicción que se espera sea correcta o menos errónea.
- **Aprendizaje no Supervisado:** En estos tipos de algoritmo se lleva a cabo el modelado con una serie ejemplos que tan sólo constan con sus valores de entrada, mientras que el sistema desconoce a qué clase o qué salida debiera surgir por cada ejemplo. Esto obliga al algoritmo a tener la capacidad de reconocimiento de patrones y ser capaz de diferenciar entre los ejemplos y dividirlos en grupos, en el que la salida será el grupo al que pertenece cada ejemplo.
- **Aprendizaje semisupervisado:** Este tipo de algoritmos son sólo una combinación de los dos anteriores, tiene en cuenta tanto los ejemplos etiquetados como los no etiquetados.

■ Enfoques

Dentro de esta rama de la inteligencia artificial se pueden tomar varios enfoques a la hora de la creación de algoritmos y programas para la resolución de los problemas planteados:

- Árboles de decisión
- Reglas de asociación
- Algoritmos genéticas
- Redes neuronales artificiales
- Máquinas de vectores de soporte
- Algoritmos de agrupamiento
- Redes bayesianas
- Aprendizaje por refuerzo

3.1.2 Deep Learning

Es un conjunto de algoritmos en *machine learning* que intenta modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no-lineales múltiples.

No existe una única definición de aprendizaje profundo. En general se trata de una clase de algoritmos para aprendizaje máquina. A partir de este punto común, diferentes publicaciones se centran en un conjunto de características diferentes, por ejemplo:

- usar una cascada de capas con unidades de procesamiento no lineal para extraer y transformar características. Cada capa usa la salida de la capa anterior como entrada. Los algoritmos pueden utilizar aprendizaje supervisado o no supervisado, y las aplicaciones incluyen reconocimiento de patrones y clasificación estadística.
- estar basados en el aprendizaje (no supervisado) de múltiples niveles de características o representaciones de datos. Las características de más alto nivel se derivan de las características de nivel inferior para formar una representación jerárquica.
- formar parte del campo del aprendizaje máquina para aprender representaciones de datos.
- aprender múltiples niveles de representación que corresponden con diferentes niveles de abstracción. Estos niveles forman una jerarquía de conceptos.

Todas estas maneras de definir el aprendizaje profundo tienen en común múltiples capas de procesamiento no lineal y el aprendizaje supervisado o no supervisado de representaciones de características en cada capa. Las capas forman una jerarquía de características desde un nivel de abstracción más bajo a uno más alto.

Los algoritmos de aprendizaje profundo contrastan con los algoritmos de aprendizaje poco profundo por el número de transformaciones aplicadas a la señal mientras se propaga desde la capa de entrada a la capa de salida. Cada una de estas transformaciones incluye parámetros que se pueden entrenar como pesos y umbrales. No existe un estándar de facto para el número de transformaciones

(o capas) que convierte a un algoritmo en profundo, pero la mayoría de investigadores en el campo considera que aprendizaje profundo implica más de dos transformaciones intermedias.

3.1.3 **Servidor Web**

3.2 **Conceptos teóricos en el lado del Cliente**

3.2.1 **Funcionamiento de las librerías Text2Speech**

4. TÉCNICAS Y HERRAMIENTAS

En este apartado se indicarán las técnicas y herramientas utilizadas durante la realización del proyecto.

4.1 Técnicas de desarrollo

En esta sección se indicarán las técnicas de desarrollo utilizadas a lo largo del proyecto.

4.1.1 Metodología de desarrollo ágil

Para llevar a cabo este proyecto hemos seguido una metodología ágil de desarrollo de proyectos.

El uso de una metodología ágil de desarrollo viene justificada por el hecho de que los datos respecto a los últimos años, la inmensa mayoría de los proyectos informáticos que se desarrollaron con una metodología clásica de gestión de proyectos han fracasado o no se han llegado a terminar. Ante este problema se ha ideado una nueva e innovadora metodología de gestión de proyectos, esta es llamada metodología de desarrollo ágil.

La metodología de desarrollo ágil se tiene como objetivo el evitar el fracaso de los proyectos, para ello se pretende que los proyectos sigan un nuevo método de desarrollo llamado *Scrum*. Las principales características del *Scrum* son:

- Adopta una estrategia de desarrollo incremental en contra partida a la ejecución completa del producto que se da en la gestión clásica de proyectos.
- Basa la calidad del resultado más en el conocimiento de las personas en equipos autoorganizados, que en la calidad de los procesos empleados a lo largo del proyecto.
- En el desarrollo ágil podemos observar un claro solapamiento de las fases de desarrollo, mientras que en la gestión clásica estas se producen de manera secuencial o en cascada.

Al usarse la metodología de desarrollo *Scrum* podemos dividir el proceso en varias partes:

- En primer lugar el desarrollo ha sido guiado a través de iteraciones, que han sido delimitadas como *Sprints*. Estos *Spints* tienen una duración de entre 1 o 2 semanas.
- En cada *Sprint* se propone una serie de tareas y objetivos a completar antes de la finalización del mismo.
- Al final de cada *Sprint* se ha realizado una reunión en la que se establecía los objetivos que se han cumplido en ese *Sprint*, esto ha sido reflejado en la planificación del proyecto

como *Retrospective Meeting*; seguidamente se proponía los nuevos objetivos para el siguiente *Sprint*, esto esta representado en la planificación del proyecto como *Sprint planning*.

Creemos que gracias al uso de este tipo de metodología el proyecto tiene mayores posibilidades de terminar de manera exitosa.

4.1.2 Desarrollo Software con control de versiones

El control de versiones en desarrollo software se puede describir como la gestión de los cambios que se producen en nuestro software o en la configuración del mismo. Entendiéndose como versión el estado en el que se encuentra el software en un determinado momento.

En el proyecto se ha usado el control de versiones sobre el software a programar por parte del alumno. Para realizar el control de versiones se ha creado un repositorio público en GitHub.

En GitHub las versiones se van determinando a través de *commits*. Lo *commits* son una actualización del estado actual del proyecto, estas actualizaciones llevan un título y un comentario para poder identificar los cambios del software y qué documentos han sido añadidos en qué *commit*.

En el proyecto la asiduidad de los *commits* no sigue ningún procedimiento estricto, sino que cuando se determinará que se ha producido un cambio en el software lo suficientemente importante se realizaría un *commit* con un comentario descriptivo del cambio que se ha producido en el software.

Lo interesante del uso de este tipo de metodología junto con la herramienta es que se pueden extraer gráficos y datos que nos aportan información bastante representativa de cómo ha ido evolucionando el proyecto en el aspecto software. También nos asegura que los cambios realizados no se pierden en caso de que la máquina falle o en caso de habernos equivocado al hacer un *commit* este puede ser deshecho. En conclusión el uso de control de versiones aporta muchas ventajas y datos con los que posteriormente podremos analizar más en profundidad la evolución del proyecto y usarlo como *feedback* para posteriores proyectos que se vayan a realizar.

4.2 Herramientas utilizadas

En este apartado se mostrará las distintas herramientas utilizadas para el desarrollo del proyecto.

4.2.1 Gestor de Tareas: VersionOne

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- PivotalTracker
- FogBugz

- VersionOne

Se ha optado por la herramienta VersionOne, que ofrece unas condiciones notablemente mejores a las otras en su versión gratuita y además resulta bastante intuitiva y fácil de usar.

Con esta herramienta nos encargaremos de generar los Sprints del proyecto y se gestionará las tareas dentro de cada uno. Esta herramienta es usada en el desarrollo ágil, que es el tipo de desarrollo que se llevará acabo en el proyecto, además de que con la herramienta podemos, posteriormente, generar una serie de gráficos e informes que nos serán de gran ayuda a la hora de documentar el proyecto y de gestionar el avance del mismo.

4.2.2 Gestor de Versiones: Git Hub

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- GitHub
- Bitbucket
- Sourcefroge

Finalmente se decidió que se iba a usar la herramienta GitHub porque se tenía experiencia previa en el uso de la misma, ofrece unas condiciones bastante razonables en su versión gratuita y se puede hace un buen seguimiento del proyecto con ella. Además de que es bastante sencilla la generación de commits en Windows porque dispone de una aplicación con la que lo commits se realizan de manera sencilla. Se puede ver de una manera bastante clara cómo ha ido avanzando el proyecto y se puede extraer unos gráficos muy útiles a la hora de documentar y determinar cómo ha avanzado el proyecto. Además al estar en su versión gratuita los miembros del jurado y cualquier persona que desee el acceso al proyecto sólo necesitará el link del mismo para poder verlo y comprobar la asiduidad de los commits y cómo ha sido la evolución del proyecto.

4.2.3 IDEL de desarrollo: Android Studio

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- Eclipse
- Android Studio

La elección de Android Studio ha sido porque no sólo es una herramienta exclusivamente dedicada a aplicaciones Android, sino que resultaba más prometedora que Eclipse; la cuál pensamos que puede quedar obsoleta para este tipo de aplicaciones. También vemos que Android Studio ofrece un gestor de instalación de paquetes de kit de desarrollo y plugins, lo cuál puede resultar muy útil a la hora de la configuración del entorno de desarrollo. Además en Android Studio no puede sólo usarse máquinas virtuales de móviles, sino que puedes conectar un dispositivo móvil al ordenador e ir ejecutando tu aplicación sobre el contando con un debugger y un logcat para errores, lo cual facilita en gran medida la programación.

4.2.4 Herramienta de Generación de Documentación: LaTeX

Se ha elegido esta herramienta debido a su facilidad de uso y a que optimiza automáticamente la estructura del producto final para ofrecer el mejor resultado visual. También prevemos que el uso de esta herramienta facilitará en gran medida la carga de trabajo al realizar la documentación lo que permitirá ahorrarse tiempo en este apartado, producir un resultado mas eficiente y bueno y, finalmente, se podrá usar el tiempo ahorrado en otras cosas para el avance del proyecto.

4.2.5 Herramientas de Deep Learning: NeuralTalk

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- Lib CCV
- Overfeat
- Deep Belief SDK

Esta herramienta ha sido elegida porque venía en el mismo idioma en el que se iba a programar el servidor, además de que su aplicación en nuestro proyecto era mucho mas práctico, pues devuelve una frase descriptiva de lo que en una imagen hay. Lamentablemente su funcionamiento e instalación son algo enrevesadas y requiere de una configuración que depende mucho de la máquina en la que se trabaje y de la estructura de directorios que esta tenga, pero una vez configurada su funcionamiento es el esperado.

4.2.6 Herramientas de Deep Learning: Caffe

Esta herramienta es usada en nuestro proyecto debido a que la herramienta NeuralTalk tiene dependencia de este proyecto para extraer características de las imágenes y luego poder realizar una predicción con ellas. Además este proyecto está bien estructurado y en si se sigue correctamente su documentación es relativamente fácil de instalar. Caffe es un arquitectura para el *deep learning* que está programada en C++ puro y en CUDA, pero también se puede usar en línea de comandos en un sistema Unix y cuenta con *wrappers* para Python y MATLAB. Sus principales características que lo convierten en un proyecto que destaca sobre los demás son:

- Es una arquitectura que funciona de manera especialmente rápida.
- Su código está bien testeado.
- Tiene buenas herramientas, modelos de referencia, demos y repositorios.
- Posibilidad de ejecutarlo tanto en CPU como en GPU

■ Anatomía de un modelo Caffe

La *deep networks* o redes profundas son modelos compositivos que se representand e forma natural como un conjunto o una colección de capas interconectadas que trabajan sobre fragmentos de datos. Caffe define una red capa a capa en su propio esquema de modelo. La red define el modelo entero desde abajo hasta arriba a partir de unos datos de entrada. Como los datos y

sus derivados fluyen a través de la red en el *Forward and Backward*, Caffe guarda, comunica y manipula la información como burbujas: la burbuja es una matriz estándar y una interfaz de memoria unificada para el *framework*. La capa que sigue es tratada como la base tanto del modelo como de la computación. La red se considera como un conjunto de capas y de sus intercionexiones. Los detalles dentro de cada burbuja describen como la información será guardada y cómo esta será enviada a través de las distintas capas y redes.

La forma en que se resuelve la predicción es configurada a parte para desacoplar el modelo de la optimización del mismo.

4.2.7 Herramientas de programación: MATLAB

Tuvimos que instalar MATLAB porque NeuralTalk usa un script de MATLAB para poder preparar las imágenes para extraerles las características, además se usa el wrapper de caffe para MATLAB.

4.2.8 Herramientas de desarrollo de servidores: Flask

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- Axis2/C
- GSOAP
- Tomcat

Se ha escogido esta herramienta en concreto porque sobre todas las demás su funcionamiento era muy inmediato y además se escribe en Python, que es un idioma muy versátil y fácil de usar. El hecho de que esta herramienta tenga un funcionamiento y programación tan sencilla la hace una herramienta que, a nuestro parecer, destaca sobre el resto y es interesante trabajar con ella. Además tiene una documentación sencilla, repleta de ejemplos y explicaciones para poder realizar tu servidor. Sus ejemplos son muy útiles y funcionan a la primera, sin tener que configurar casi nada. Tiene una forma de establecer los URLs de forma muy sencilla y es fácil estructurar tu servidor con esta API. A pesar de ser una API tan fácil de usar tiene una gran potencia y se pueden construir con ella servidores bastante grandes y fácilmente escalables, por tanto se convierte en la herramienta perfecta para la programación de nuestro servidor.

4.2.9 Librería Text2Speech para Android

4.2.10 Librería de Apache para conexiones Http para Android

4.2.11 Librería de Google para la traducción de texto

5. ESTADO DEL ARTE

En este apartado...

6. ASPECTOS RELEVANTES DEL DESARROLLO DEL PROYECTO

En este apartado se introducen los aspectos más relevantes del proyecto.

6.1 Dificultades encontradas

Durante el desarrollo del proyecto nos hemos encontrado varias dificultades que han hecho que el proyecto se retrase considerablemente y su avance no haya sido ni fácil ni rápido.

6.1.1 Dificultades con GSOAP y Apache

Este fallo es el que más ha retrasado al proyecto y ha supuesto una dificultad enorme a la hora de llevar a cabo el mismo.

Empezamos con que para usar GSOAP se tuvo que estudiar una serie de cosas para poder adquirir los conocimientos necesarios para usar la herramienta, dichos conocimientos serán listados aquí:

- XML: Se tuvo que coger un nivel adecuado en el uso de XML ya que la herramienta GSOAP se basa en el uso de este tipo de archivos como medio de comunicación en las distintas peticiones y respuestas que procesa. Además el XML también es necesario para comprender el funcionamiento de SOAP y de WSDL, los cuales son completamente necesarios para entender el funcionamiento de la herramienta GSOAP.
- SOAP: Esta especificación se tuvo que estudiar para comprender el funcionamiento de la herramienta GSOAP y en qué se basaba su funcionamiento, entender el por qué debía funcionar la herramienta y como se realiza la comunicación gracias a ella. Aunque el SOAP no es usado directamente cuando usas GSOAP es necesario conocer esta especificación ya que GSOAP sí que usa WSDL, para el cual tenemos que tener un conocimiento básico, al menos, de SOAP para poder usarlo.
- WSDL: Esta otra especificación sí que se usa directamente en la herramienta GSOAP y básicamente con ella vertebras toda la aplicación que vas a hacer, de hecho tienes dos opciones:

  La primera es usar un archivo WSDL donde especificas las operaciones que el servidor va a realizar, después con la herramienta GSOAP generas todos los stubs y documentos necesarios para hacer tu aplicación.

  La segunda sería a través de un documento de tipo .h o una cabecera de C. Con el cuál también generas los stubs y documentos necesarios para programar tu servidor, entre dichos documentos se econtrará un archivo WSDL que contendrá la especificación de las

operaciones que hay dentro del fichero cabecera que hayas usado. Pero incluso en esta opción necesitas entender SOAP y WSDL porque a través de comentarios tienes que especificar características que irán directamente al fichero WSDL, y que serán necesarios para el correcto funcionamiento del servidor.

Una vez se ha estudiado lo anterior se paso al estudio de la documentación de la herramienta GSOAP, además de el intento de hacer que funcionen sus ejemplos.Cuando se consiguió que funcionarán sus ejemplos se paso a la programación de un servidor propio, una vez se programo y se hicieron las pruebas de que estaba bien programado se procedió a intentar que este funcionará desde un cliente GSOAP. Para que funcionará con el cliente GSOAP se hizo una investigación de cómo hacer que el servidor funcionará en localhost y, siguiendo la recomendación que en la documentación de GSOAP encontramos, se instaló Apache y se inteto usar el módulo de Apache para su funcionamiento con GSOAP.

Como conclusión sacamos que, tras una larga investigación y mucho tiempo dedicada a esta herramienta, esta herramienta no tenia la documentación suficiente como para poder hacerla funcionar con Apache y, a pesar de haberlo intentado muchas veces, no conseguimos que el servidor GSOAP programado por nosotros devolviera alguna vez un resultado coherente al cliente. Finalmente desechamos la opción de trabajar con esta herramienta y le dimos un giro al proyecto con el que esperábamos tener avances más rápidos y mejores, optamos por la programación de un servidor en Flask.

6.1.2 Dificultades con DeepBeliefSDK

Al final no se pudo usar.

6.1.3 Dificultades en la instalación de Herramientas

Caffe tuvimos problemas de compatibilidad con open-cv Neuraltalk tuvimos problemas con la cantidad de dependencias que tenía.

6.1.4 Dificultades con NeuralTalk

Tuvimos problemas con la herramienta de MATLAB al ejecutar un script desde el servidor.

6.1.5 Dificultades con Android

Problemas al conectarnos con el servidor.

7. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

En este apartado se presentarán las conclusiones y las posibles líneas de futuro que saquemos del proyecto.

7.1 Conclusiones del Proyecto

7.2 Lineas de trabajo Futura

I. PLAN DEL PROYECTO SOFTWARE

I.1 Plan de Proyecto Software

Este apartado está dedicado al plan de proyecto software donde se comenta todo el proceso de planificación del proyecto.

I.1.1 Introducción

La planificación del proyecto se lleva a cabo de forma ágil, con la metodología SCRUM. Y la plataforma sobre la que se ha trabajado para realizar la planificación es VersionOne. Y la planificación se ha dividido en *sprints* o iteraciones, las cuáles tienen la mayoría una duración de una semana, aunque por problemas algunas han durado más.

■ Problemas encontrados

Debido a que las primeras dos iteraciones se realizan de forma simulada, osea después del tiempo previsto, por eso no se pueden sacar sus gráficos *burn-down*. Además de que al día 30 de uso del VersionOne se caducó la prueba y perdí todos los datos del proyecto, pero contactando con el soporte de VersionOne estos me reabrieron la prueba por una semana para extraer los datos necesarios, pero debido a la tardanza en su respuesta hay un *sprint* que dura 3 semanas y no podemos extraer su gráfico *burn-down* tampoco.

I.1.2 Planificación temporal del proyecto

En este apartado se presentan los distintos *sprints* y las tareas de cada uno.

■ Sprint 1:23 de Diciembre al 12 de Febrero

Este *sprint* es de una duración bastante mayor al resto porque se empezó con mucha antelación y su objetivo era el de documentarse. Se estudió los distintos aspectos del proyecto y se tomo contacto con el mismo.

Este *sprint* es importante porque es donde se asientan las bases del proyecto y se empieza a tomar contacto con el proyecto. Sin este *sprint* se hubiera empezado sin ninguna base y el comienzo del proyecto hubiera sido más tardío y , por tanto, su avance más lento.

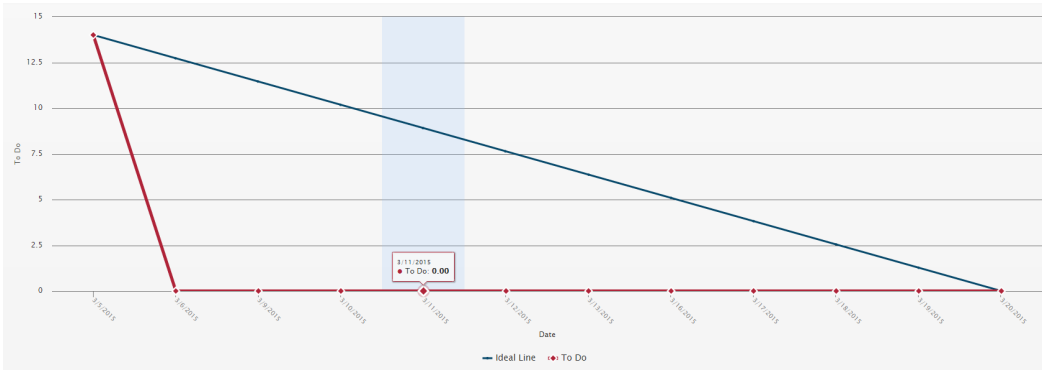

Figura I.1.: Burn-down del sprint 3

■ Sprint 2:12 de Febrero al 26 de Febrero

Esta iteración se dedicó exclusivamente a la decisión de las herramientas que se van a usar en el proyecto y a la familiarización con las mismas.

Aquí se decidió:

- Gestor de versiones: GitHub
- Gestor de Tareas: VersionOne
- Entorno de desarrollo: Android Studio
- Herramienta de ofimática: LaTeXcon su editor TeXMaker

■ Sprint 3:26 de Febrero al 6 de Marzo

En este *sprint* se instalaron y se crearon cuentas de las herramientas del anterior *sprint*. Se dividió en distintas tareas:

- Crear cuenta VersionOne
- Crear cuenta en GitHub
- Instalar Android Studio
- Instalar librería DeepBeliefSDK

Además se procedió a invitar a los tutores a VersionOne, realizar el primer commit en GitHub, probar ejemplos de Android y del DeepBeliefSDK y, finalmente, se generó la documentación asociada a este *sprint*.

Podemos ver el gráfico *burn-down* de este sprint en la imagen I.1.

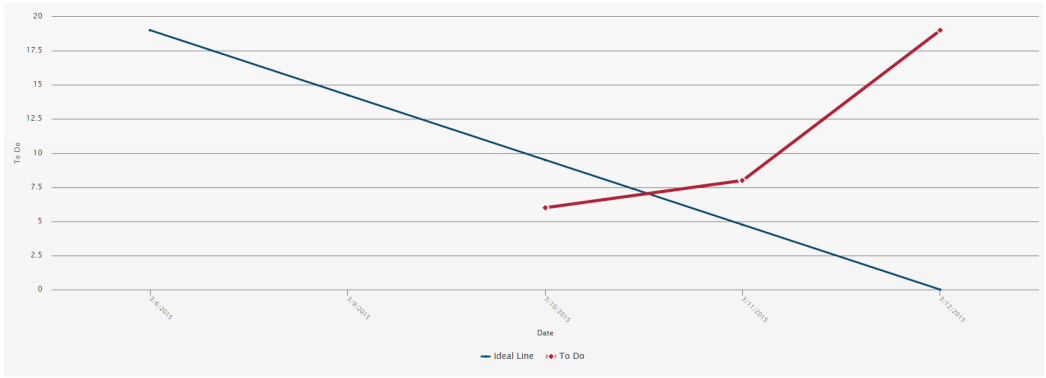

Figura I.2.: Burn-down del sprint 4

■ Sprint 4:6 de Marzo al 13 de Marzo

En esta iteración se profundiza, sobretodo, en la utilización de la librería DeepBeliefSDK, se probarán ejemplos y trabajarán sobre ellos. Además se generará la documentación asociada al *sprint*.

Se identifican las tareas:

- Consulta de la documentación de Android: se pasa a avanzar en la programación de Android y se empiezan a realizar las primeras aplicaciones de prueba con ayuda de la documentación de Android.
- Instalación Linux: Se procede a instalar un Linux con su distribución Ubuntu en una máquina virtual sobre la que trabajar.
- Impresiones de Pantalla: Se hacen capturas de pantalla del proceso de instalación de las herramientas para la futura documentación en la sección: Manual del programador (II)

Este fue el primer *sprint* en el que añadimos el *Retrospective Meeting* y el *Sprint Planning* al VersionOne, pese a que sí que habíamos hecho estas reuniones.

En el *Retrospective Meeting* se determino que:

- Se estudió el funcionamiento de la librería DeepBeliefSDK
- Se hicieron las impresiones de pantalla para la documentación
- Se hicieron más pruebas con Android.
- Se envió invitaciones a los profesores a VersionOne y Github

El *Sprint Planning* simplemente sirvió para determinar las tareas a realizar en esta iteración, las cuáles ya han sido comentadas antes.

Además esta iteración viene acompañada con un gráfico *burn-down*s el cuál puede verse en la imagen I.2

I.1.3 Sprint 5:13 de Marzo al 27 de Marzo

Este *sprint* está dedicado en su mayoría al aprendizaje de GSOAP, que es un *framework* para poder programar servidores en código C y C. El estudió de esta herramienta llevó bastante tiempo debido a que su documentación, no solo estaba puramente en inglés, sino que además no era lo suficientemente precisa como para alguien que estuviera empezando a programar servicios WEB.

En el *Sprint Planning* se propuso como objetivos a tener en el siguiente *sprint*:

- Evaluar si usar Axiss2/c o GSOAP para el servidor
- Continuar con la documentación
- Ejemplos de Android

En este *sprint* podemos identificar las siguientes tareas que finalmente se definieron en la herramienta VersionOne:

- Pruebas en Linux: Se hacen pruebas de la librería GSOAP en la máquina virtual de Linux, en su distribución Ubuntu.
- Pruebas en Android: Se hacen pruebas de distintos códigos y aplicaciones en Android que posteriormente nos puedan servir
- Primera versión del Cliente: Se genera la primera versión del cliente Android para el proyecto. Esta sólo es una interfaz sencilla que toma una foto con el móvil y la guarda.
- Estudio de WSDL y SOAP: Se estudia estas dos especificaciones de XML, junto con un pequeño repaso de XML. Estas dos especificaciones son totalmente necesarias para el desarrollo de la aplicación, aunque al final puede que no se lleguen a usar directamente, el conocimiento de las mismas es requisito indispensable para conocer cómo trabaja el servidor internamente.
- Descargar materiales: Para el estudio y trabajo con la librería GSOAP es necesario una serie de archivos, esta tarea es en la que nos descargamos todos los necesarios.
- Primera versión del fichero WSDL: Un versión inicial de un fichero WSDL con el que generar archivos *stubs* con la herramienta GSOAP. Este fichero contiene las especificaciones de los servicios que dará el servidor al cliente y qué tipo de datos admite y devuelve.
- Generar documentación:Como en todas las iteraciones se procura generar la documentación asociada.

En el *Retrospective Meeting* se determina qué tareas del *sprint* han sido terminadas, cuáles no y el avance de la iteración. En este caso se identificó lo siguiente:

- Se decidió usar GSOAP porque era más sencillo.
- Se estudió GSOAP y se consiguió implementar un cliente en Linux de un ejemplo básico de calculadora.

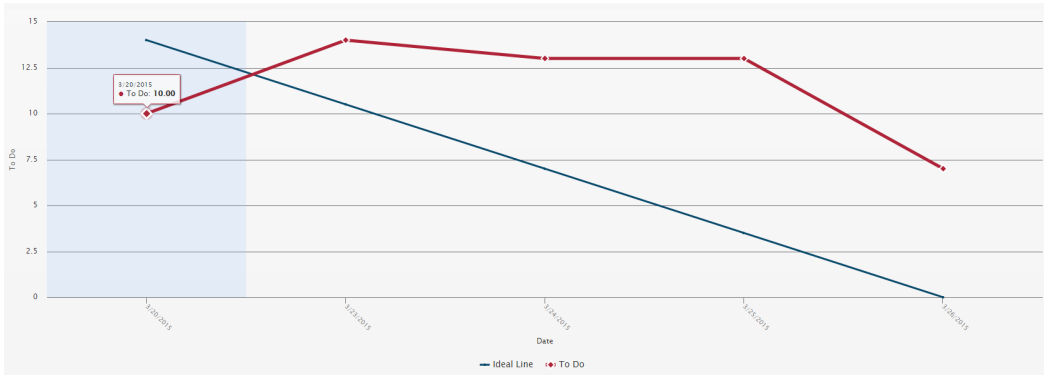

Figura I.3.: Burn-down del sprint 5

- Se continuó con ejemplos de Android creando el proyecto del cliente en su primera versión, aunque este no hacía nada más que mostrar una imagen por pantalla.

Esta iteración también tiene un gráfico de tipo *burn-down* asociado y se puede ver en la imagen **??**

■ Sprint 6: 28 de Marzo al 22 de Abril

En esta iteración se procedió a crear prototipos, tanto de cliente como de servidor, del proyecto para poder empezar con el desarrollo software del mismo.

En la planificación dentro del *Sprint Planning* se determinó:

- Tratar de implementar el ejemplo básico de servidor, con el que podamos hacer las primeras pruebas.
- Tratar de hacer el cliente en Android, que se conectara al ejemplo básico de servido. Con esto empezaremos a establecer cómo se realizará la conexión entre cliente y servidor.
- Tratar de hacer el prototipo con DeepBeliefSDK para empezar a intentar conectarlo con el servidor.
- Continuar con la documentación.

Esos fueron los objetivos fijados para este *sprint*, que es inusualmente largo debido a la serie de problemas que se encontraron en el mismo y que son explicados enn el *Retrospective Meeting*.

Mientras que en VersionOne lo que tenemos definido es:

- Instalar Apache y demás herramientas: Nos encontramos con que el servidor de GSOAP necesitaba de Apache para poder ejecutarlo en localhost y así establecer la conexión con el cliente Android, así que procedimos a la instalación de las herramientas necesarias.
- Construcción del servidor: Se montó el servidor con un ejemplo de calculadora básico y se comprobó con las propias herramientas de GSOAP, que este estaba bien definido.

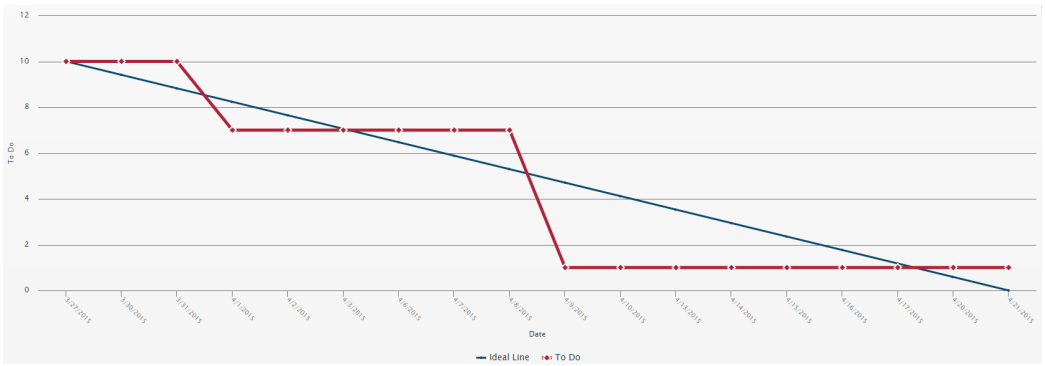

Figura I.4.: Burn-down del sprint 6

- Construcción del cliente: Se montó un cliente GSOAP con el que también se probo el ejemplo básico de servidor, este y el servidor funcionaban de manera adecuada.
- Pruebas con distintos servicios en Linux: Se remontaron varias veces el cliente y el servidor, pero con distintas especificaciones SOAP, para probar varios tipos de datos en el envío.
- Primera versión del Servidor: Después de construir el ejemplo básico se paso a conectar el DeepBeliefSDK con el servidor y este con Apache. Se empezó por probar la conexión con Apache, la cuál resultó un rotundo fracaso.
- Escritura de la documentación: Se genera, como siempre, la documentación asociada a la iteración.

Finalmente tenemos nuestro *Retrospective Meeting*, en el que determinamos lo siguiente:

- No se consiguió conectar GSOAP con Apache
- La documentación para el proceso de conectar Apache y GSOAP es escasa y la poca que hay no nos enseña un procedimiento correcto para hacer funcionar esto.
- Se decide abandonar la programación en C y la librería DeepBeliefSDK
- Conclusión: No se consiguieron los objetivos y se decide tomar un nuevo rumbo con el proyecto.

Podemos observar el gráfico de la imagen I.2, en el que observamos el gráfico *burn-down* del *sprint*.

II. MANUAL DEL PROGRAMADOR

II.1 Manual del programador

En esta sección se procederá a la explicación detallada de cómo instalar las herramientas necesarias y qué herramientas son necesarias para trabajar sobre este proyecto.

II.1.1 Instalación del JDK

La primera, y más esencial de las herramientas, es el JDK de java, que es el set o conjunto de herramientas y librerías para los desarolladores de java.

Primero deberemos ir a la página de Oracle en la que descargaremos el jdk, la página debería tener un aspecto más o menos como el de la imagen II.1 En dicha página tendremos que aceptar la licencia y posteriormente descargar el JDK que sirva para nuestra máquina. Una vez hemos descargado dicho archivo, lo ejecutamos. Una vez ejecutado seguimos los siguientes pasos para su instalación.
En la imagen II.2 vemos el primer paso para la instalación del *JDK*, el cual consta de dos pantallas. La primera será solamente una pantalla de bienvenida, por lo que debemos pulsar a *next* o siguiente. La segunda pantalla nos muestra los elementos que van a ser instalados, esto no lo tocamos; además nos muestra también el directorio en el que queremos instalar el *JDK*, esto lo podemos dejar como está o podemos, si queremos, configurarlo en un directorio personal que queramos. Lo único es que habrá que tener cuidado con el *PATH* del equipo para que luego *Android Studio* pueda encontrar la distribución de *JDK* que tengamos en nuestra máquina.

En la imagen II.3 nos encontramos el paso 2 para la instalación de nuestro *JDK*, este también consta de dos pantallas. La primera es el estado de la instalación, veremos una barra que se irá llenando en función de que la instalación vaya avanzando. Finalmente se nos mostrará la pantalla que nos indicará que la instalación se ha realizado correctamente y nos da la opción de pulsar el botón *Next Steps* uqe nos llevará a la *API de Java* o simplemente finalizar la instalación.

II.1.2 Instalación de Anrdoid Studio

Para la programación del cliente se usará *Android Studio*, una herramienta ideada para programar en *Android* exclusivamente. Lo primero que debemos hacer es dirigirnos a la página web donde descargaremos Android Studio, la cual tendrá un aspecto similar al de la II.4. En ella haremos *click* sobre el botón de descarga (*Download Android Studio*) y procederemos a la instalación ejecutando el fichero que hemos descargado, todo este proceso es para un sistema operativo *Windows*.

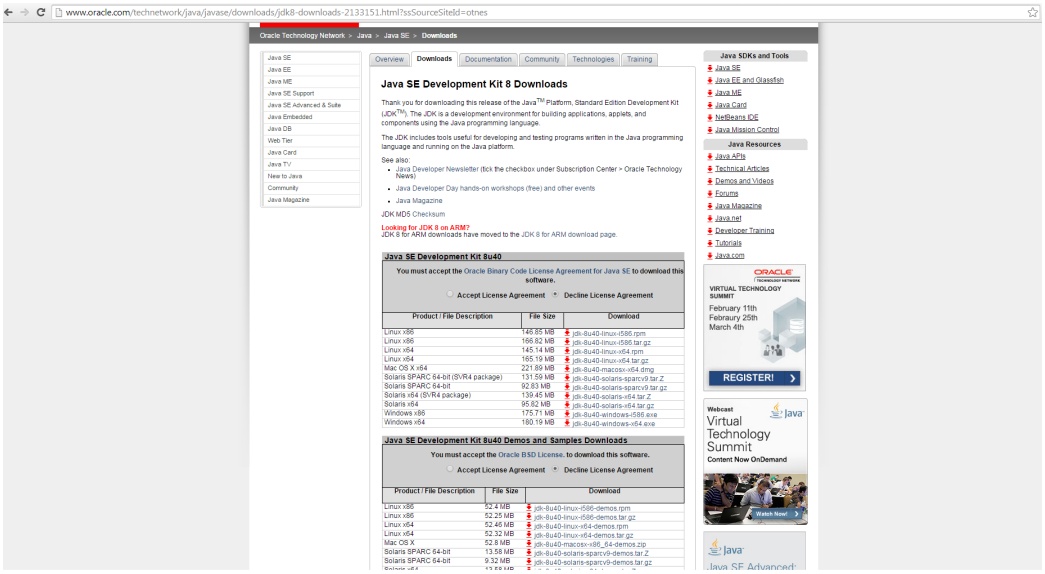

Figura II.1.: Página de descarga del JDK de Java

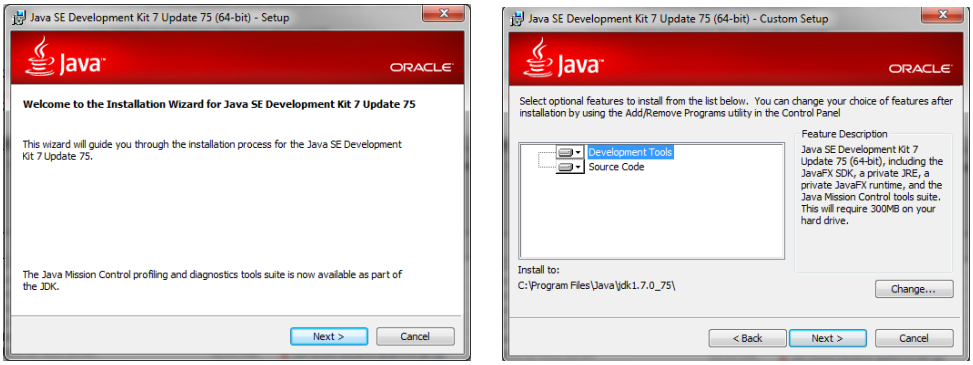

Figura II.2.: Instalación del JDK, paso 1.

Figura II.3.: Instalación del JDK, paso 2.

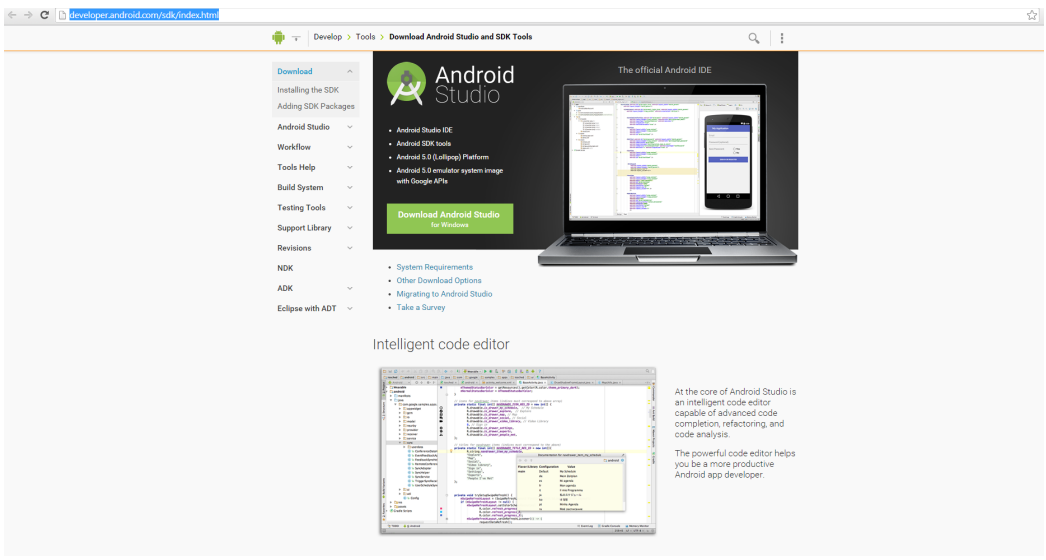

Figura II.4.: Página de descarga de Android Studio

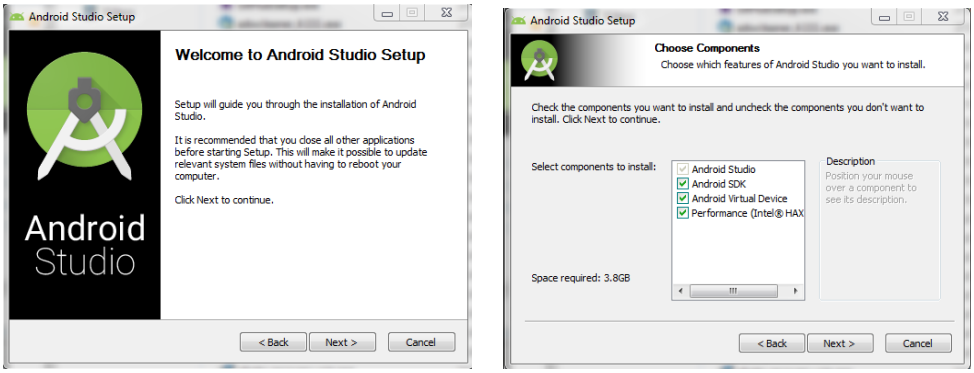

Figura II.5.: Instalación de Android Studio, paso 1.

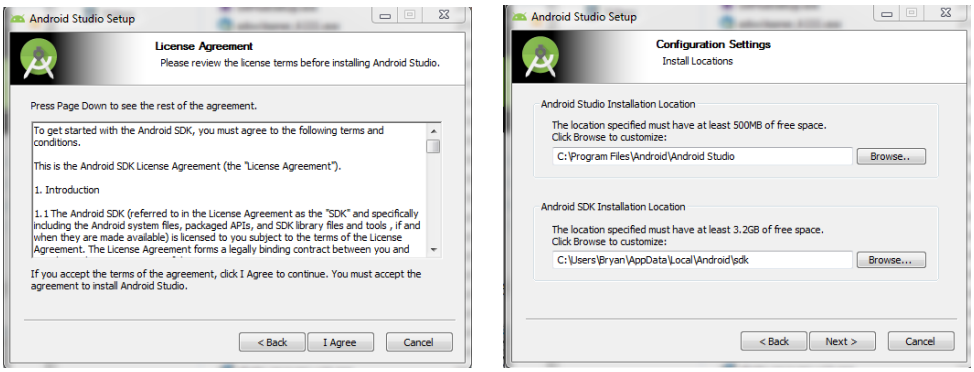

Figura II.6.: Instalación de Android Studio, paso 2.

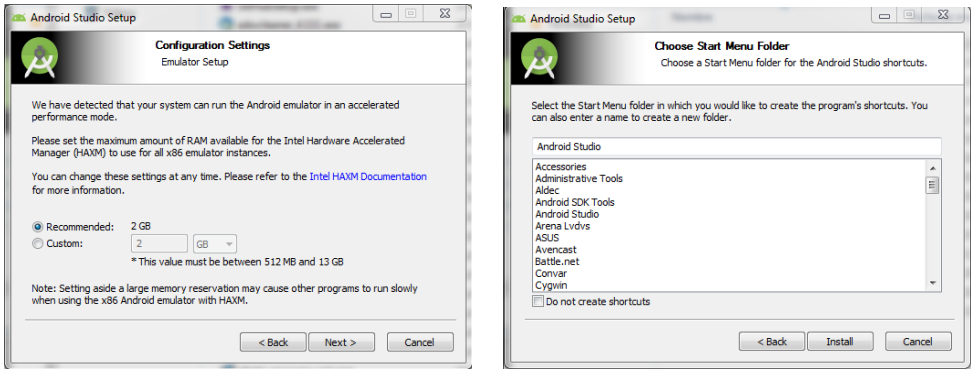

Figura II.7.: Instalación de Android Studio, paso 3.

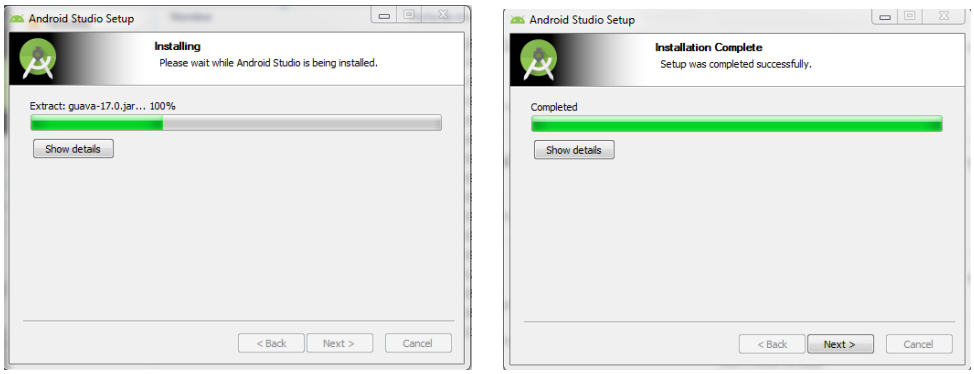

Figura II.8.: Instalación de Android Studio, paso 4.

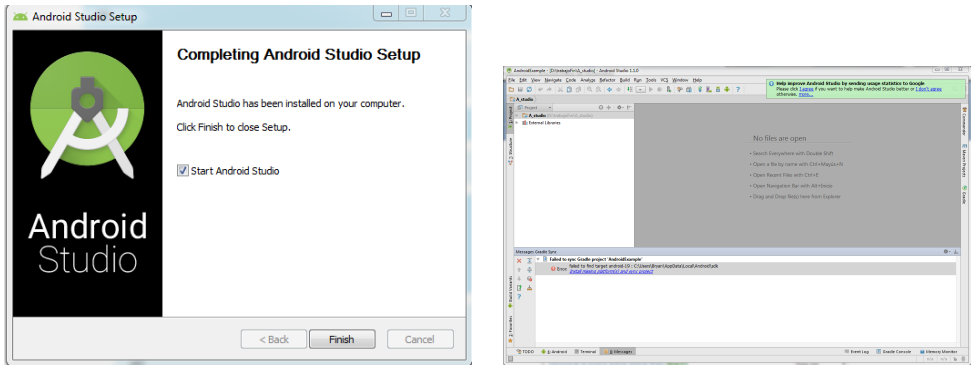

Figura II.9.: Instalación de Android Studio, paso 5.

En el primer paso de la instalación, que se puede observar en la imagen II.5, nos encontraremos con dos pantallas. La primeara es simplemente una pantalla de bienvenida a la instalación de nuestro *Adroid Studio*, debemos pulsar al botón *next* para empezar nuestra instalación. Entonces pasamos a la siguiente pantalla y nos da la opción de elegir qué elementos instalar, nosotros hemos dejado todos marcados pero puede que algunos no sean obligatoriamente necesarios. Una vez seleccionados los componentes de nuestra instalación pasamos a dar el botón *next* de la pantalla, entonces saltamos al paso 2.

El segundo paso de nuestra instalación lo podemos observar en la imagen II.6. Una vez hemos pasado la selección de componentes en nuestro *Android Studio*, tenemos que aceptar la licencia del software, pasaremos a leer la licencia, si es de nuestro interés, y cuando estemos de acuerdo con ella, pulsaremos el botón *I Agree*. Esto no habrá llevado a la siguiente pantalla de la instalación, en la que podremos elegir los directorios en los que instalaremos nuestros componentes, nosotros los hemos dejado por defecto aunque se pueden personalizar en función de las necesidades del usuario. Una vez establecida la configuración de directorios en la instalación, pulsaremos el botón *next*.

Pasaremos entonces al tercer paso de la instalación, la cual puede ser vista en la imagen II.7. En la primera pantalla de este paso tendremos que elegir la cantidad de memoria *RAM* que asignaremos a nuestro *Android Studio*, podemos dejar la cantidad recomendada o podemos asignarle más cantidad, si disponemos de ella, para que tenga un funcionamiento más fluido. Una vez establecido esto, se pulsara el botón *next*. En nuestra segunda pantalla nos pregunta el directorio en el que se iniciará la aplicación y puedes poner la opción de no crear un acceso directo, nosotros hemos dejado la configuración por defecto. Finalmente pulsamos el botón *Install*.

Ahora en el paso 4, el cual está representado en la imagen II.8, solamente tendremos que observar cómo avanza la instalación, esto se nos irá mostrando a través de una barra de progreso. Una vez se la barra se ha llenado podremos hacer *click* en el botón *Next* para pasar al último paso de nuestra instalación.

En nuestro quinto y último paso, el cual podemos ver en la imagen II.9, se nos informará de que la instalación ha terminado y tendremos la opción de iniciar nuestro *Android Studio* nada más acabar la instalación. Entonces pulsamos al botón *Finish*, se nos abrirá nuestro *Android Studio* y con esto la instalación se considera terminada.