UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

SurveyingPointCode

Automatización del proceso de delineación a partir de datos de un Levantamiento Topográfico.

Presentado por
José Eduardo Risco Sánchez-Cortés
en Universidad de Burgos — 15 de mayo de 2019
Tutores: Dr. César García-Osorio
y Dr. Carlos López Nozal

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado se va a detallar como se ha llevado a acabo la planificación del proyecto. En la planificación es donde se estudia el coste tanto en tiempo, como en recursos y también, aunque sea un proyecto educacional, el coste económico y sus posibles beneficios.

Con los resultados obtenidos realizaremos la planificación temporal del proyecto y el estudio de viabilidad del mismo.

A.2. Planificación temporal

Como ya se ha mencionado en el inicio del proyecto se ha optado por seguir la metodología ágil *Scrum*. Se ha intentado seguir fielmente, con la salvedad de que normalmente en este tipo de proyectos participan varias personas y en este ha participado solo una persona, con la supervisión del tutor.

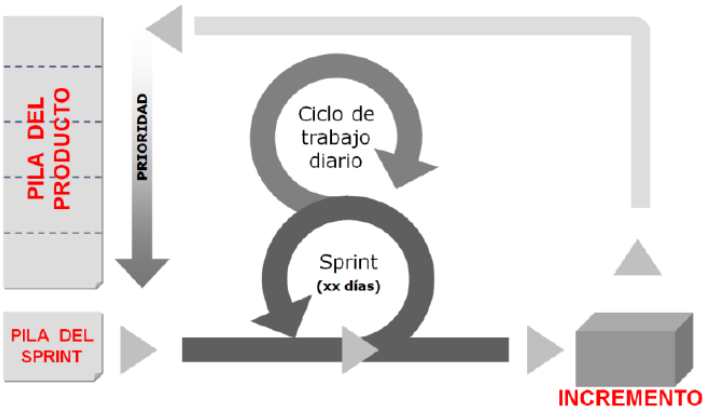

Figura A.1: Diagrama del ciclo iterativo *Scrum*.

Scrum se basa en aplicar una estrategia de desarrollo incremental, que intenta mantener un ritmo de avance constante. Se planifican una serie de *sprints* donde se debían desarrollar una serie de tareas completamente funcionales. La duración de estos *sprints* era de una semana y al final de cada *sprint* se realizaban reuniones para comprobar si se habían cumplido los objetivos y planificar el siguiente *sprint*.

Los requisitos del sistema se registraron en dos formatos:

- **Pila del producto:** como una lista ordenada de todo aquello que el propietario de producto cree que necesita el producto. La pila del producto nunca se da por completada; está en continuo crecimiento y evolución.

- **Pila del *sprint*:** como una lista de las tareas necesarias para construir las historias de usuario que se van a realizar en un *sprint*. Refleja los requisitos vistos desde el punto de vista del equipo de desarrollo.

Para la gestión del proyecto se utilizó *ZenHub*, en ella podíamos hacer el seguimiento de lo que está en revisión, lo que debe probarse, lo que se está haciendo o lo ya cerrado.

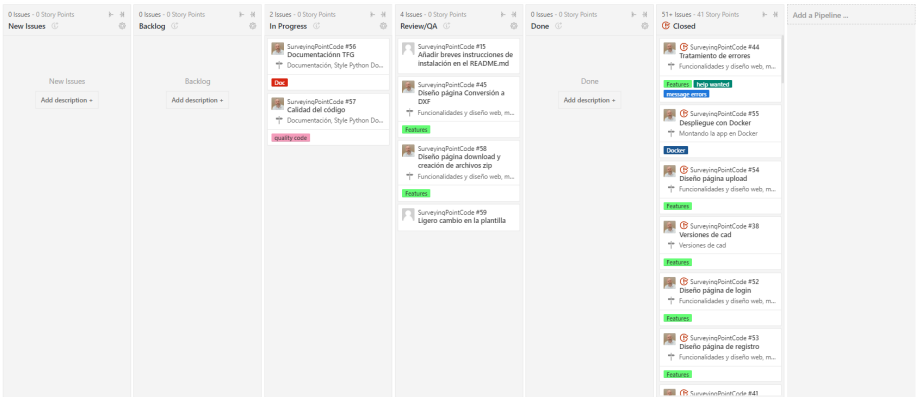

Figura A.2: Gestión de tareas con *ZenHub* en una fase del proyecto.

A continuación se describen los diferentes *sprints* que se han realizado:

Sprint 1 (24/01/2019 - 30/01/2019)

En este primer *sprint* y como primera toma de contacto con el proyecto, los objetivos planteados fueron los siguientes:

- Preparación del entorno de trabajo. El proyecto se iba a desarrollar en Python, se instaló la version *Python 3.7* y como entorno de la aplicación *PyCharm 2019.1.1 Comunity Edition*.
- Repaso de conocimientos en *GIT* y creación del repositorio en *GitHub*. Se refrescaron los conocimientos en *GIT*, en algunos tutoriales online como *git - la guía sencilla* [3] y *Tutorial de Git. Manual básico con ejemplos* [6]
- Formación en *PEPs*. Para conocer bien las guías de estilo y la convenciones de *Python*, se consultaron las guías PEP 8 y PEP 257, mencionadas en la memoria.
- Formación en LaTeX. Se consulto como guía principal, el libro Edición de Textos Científicos LaTEX[1]. Y como herramientas para realizar la memoria se instalaron las aplicaciones :*Texmaker 5.0.3* y *MikTex 2.9*.
- Diseño del logotipo de la aplicación.

Figura A.3: Logo de *SurveyingPointCode*.

Sprint 2 (31/01/2019 - 06/02/2019)

Los objetivos planteados fueron los siguientes:

- Formalización de la entrada de datos: Se procede a formalizar la gramática que debe tener el archivo de entrada y se determina que: El archivo de entrada será un archivo de texto, compuesto por una o múltiples lineas. Estas lineas serán los puntos medidos en campo y cada linea tendrá la siguiente estructura:

  número de punto, coordenada x, coordenada y, coordenada z, código

  - El numero de punto debe ser de tipo *integer*
  - Las coordenadas x,y,z de tipo *float* o *integer*
  - El código de tipo string, pudiendo estar formado por letras, números y los signos '-' y '+'

- Elección de una herramienta de análisis sintáctico: Se elige una herramienta de análisis sintáctico para poder validar los archivos de entrada. Las opciones eran:

  - *Ply*
  - *ANTLR*
  - *Flex Bison*

  Nos decantamos por *Ply* ya que está implementada completamente en *Python* y encaja perfectamente con la filosofía de realizar la mayor parte del proyecto con este lenguaje.

- Creación de un prototipo: Se creó un prototipo que permitía indicarnos si el archivo de entrada era correcto o no. Se comprobó su funcionamiento con dos archivos, uno con la gramática correcta y otro gramática incorrecta, y el resultado fue satisfactorio en ambos casos.

Sprint 3 (07/02/2019 - 13/02/2019)

Como objetivo se planteó seguir con el desarrollo de este primer prototipo, ampliando sus funcionalidades. Se definieron las siguientes tareas:

- Organizar elementos en listas de capas: Se leerán las lineas del fichero de entrada (puntos medidos en campo) y se organizarán en una estructura de datos en forma de lista , donde cada lista contendrá los elementos (puntos) correspondientes a una misma capa. Se obtuvo un resultado correcto.

- Identificar y organizar los diferentes tipos de lineas: Se pretende identificar todos los puntos que forman unas linea, y almacenar todas las lineas existentes en estructuras de datos. Se obtuvo un resultado correcto.

- Comenzando con biblioteca `ezdxf`: Se comienza a estudiar la biblioteca `ezdxf` y a realizar pequeñas pruebas, incluyendo estas en el prototipo. Dibujar las líneas y *splines* de un fichero de entrada.

  El resultado fue correcto, lo podemos ver en la siguiente imagen.

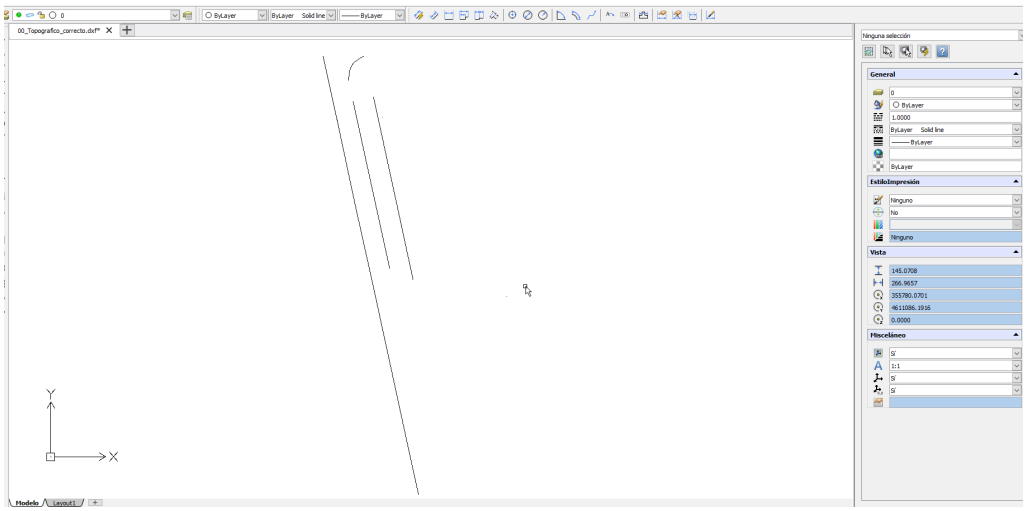

Figura A.4: Dibujo con líneas y *splines*

Sprint 4 (14/02/2019 - 20/02/2019)

Una vez acabado el primer prototipo funcional que podía validar un archivo, diferenciar las diferentes lineas y curvas, y generar un archivo DXF, el objetivo de este *sprint* fue comenzar a crear un prototipo basado en una aplicación Web, utilizando el *micro framework* `Flask`. Para ello se definieron las siguientes tareas:

- Formación en `Flask`: Se consultó la documentación de `Flask` mencionada en la memoria.
- Prototipo con `Flask`: Se creó un prototipo que permitía subir un archivo desde el equipo del usuario al servidor. Se obtuvo un resultado correcto.
- Procesado del archivo subido y descarga del archivo DXF generado: El prototipo debería validar el archivo subido por el usuario, lo convertiría a DXF, y permitiría descargarlo en su equipo. Se obtuvo un resultado correcto.

Aquí empezó a trabajar con un entorno virtual creado en *Anaconda*[1] por sugerencia del tutor.

[1] *Anaconda*: https://www.anaconda.com/

Sprint 5 (21/02/2019 - 27/02/2019)

Ya teníamos la aplicación Web funcionando, el siguiente paso sería la validación de usuarios, el uso de una base de datos y a ser posible comenzar a investigar con *Docker* y alojar la base de datos en un contenedor. Para ello se definieron las siguientes tareas:

- Comenzando con *Docker*: Se consultó la documentación de emphDocker mencionada en la memoria,y se configuró un contenedor*PostGIS* para alojar la base de datos. También se configuró otro contenedor *PgAdmin4* para poder administrar y ver las modificaciones en la base de datos más fácilmente. También serviría para probar como conectar dos contenedores entre sí.

- Validación de usuarios: Se implementó la aplicación para que un usuario pudiera logearse y registrarse. Se diseño el modelo de la base datos y la base de datos se crearía automáticamente al arrancar la aplicación. Se utilizaron las librerías `SQLAlchemy` y `Flask-Login` ,mencionadas en la memoria, de las cuales se estudió su documentación. Se obtuvo un resultado correcto.

Sprint 6 (08/03/2019 - 13/03/2019)

Este *sprint* se basó en el aprendizaje de *HTML*, formularios y *Bootstrap*. Incluyendo el tratamiento de mensajes `flash` en `Flask`.

También se estudió como incorporar un visor en el navegador, para poder visualizar el plano. Se estudiaron varias opciones:

- Convertir el archivo a formato SVG: Se convirtió el archivo DXF a SVG, con un conversor online[2], pero el resultado obtenido no era lo esperado. Se probó con otros conversores siendo el resultado igual de malo, por lo que se descartó esta opción.

- Usar el visor *Three-Dxf*: *Three-Dxf*[3] en un visor en *JavaScript*, para archivos DXF, que se podría incorporar a cualquier aplicación Web. En su documentación indicaba que soportaba los elementos que usábamos en nuestros DXF, como los tipos de lineas, tipos de símbolos, etc. Pero comprobándolo resultó que no era así, daba errores. Conseguía

[2] *convertio*: https://convertio.co/es/dxf-svg/
[3] *Three-Dxf*: https://github.com/gdsestimating/three-dxf

visualizar archivos muy simples, pero no los que necesitábamos en este proyecto.

Al final se descartó la idea de que la aplicación tuviera un visor incorporado.

Sprint 7 (14/03/2019 - 20/03/2019)

Este *sprint* se basó en implementar funciones que crearan los elementos del archivo DXF. Para ello se definieron las siguientes tareas:

- Creación de capas: Creación de capas en el DXF, a partir de los códigos del archivo de campo. Hay tres capas que se deberían crear siempre:
  - *Point:* Capa que deberá contener todos los puntos.
  - *Altitude:* Capa que deberá contener la altitud de todos los puntos. en forma de texto.
  - *Label:* Capa que deberá contener el código de todos los puntos, en forma de texto.

  Se obtuvo un resultado correcto.
- Inserción de puntos en el modelo: Se insertan todos los puntos del archivo de campo en el modelo del dibujo. Se dibujan también , los textos correspondientes al código y a la elevación del punto, cada uno en su correspondiente capa, todo ello interpretando la codificación. Se obtuvo un resultado correcto.
- Creación de circunferencias: Se crearían círculos, a partir de un punto y un radio, y se guardaría en su correspondiente capa, todo ello interpretando la codificación. Se obtuvo un resultado correcto.
- Creación de líneas: Se crearían líneas, a partir de una serie de puntos, y se guardarían en su correspondiente capa, todo ello interpretando la codificación. Se obtuvo un resultado correcto.
- Creación de cuadrados: Se crearían cuadrados, a partir de dos puntos, y se guardarían en su correspondiente capa, todo ello interpretando la codificación. El cuadrado siempre se dibujaría a la derecha de la alineación definida por estos dos puntos. Se obtuvo un resultado correcto.

- Creación de rectángulos: Se crearían rectángulos, a partir de tres puntos, y se guardarían en su correspondiente capa, todo ello interpretando la codificación. Se obtuvo un resultado correcto.

- Uso de puntos no medidos en campo: Se utilizarían puntos no medidos en campo para la generación de lineas y se guardarían en su correspondiente capa, todo ello interpretando la codificación. Se obtuvo un resultado correcto.

Una vez dibujados los elementos, se mejoró estéticamente el dibujo,modificando el tamaño de los textos y su posición respecto a los puntos.

Sprint 8 (21/03/2019 - 27/03/2019)

En este *sprint* se comenzó con la realización de test unitarios, estudiando las diferentes opciones. También como se podría leer un archivo DXF proporcionado por el usuario, que contuviera símbolos en forma de bloques y una vez , incorporarlos al modelo de dibujo que queríamos generar. Para ello se definieron las siguientes tareas:

- Estudio de una biblioteca de Python para la realización de los test unitarios : Se consideraron las bibliotecas: `unittest`[4] y `pytest`[5]. La opción elegida fue `unittest` esta biblioteca ya se había usado durante la realización del grado.

- Comienzo con los test unitarios.

- Crear una función para cargar un archivo DXF.

- Crear una función extraer los bloques del archivo DXF.

- Incorporar los bloques al modelo del dibujo.

- Asociar cada bloque con cada punto correspondiente, interpretando la codificación.

Se obtuvo un resultado correcto.

Sprint 8 (28/03/2019 - 03/04/2019)

En este *sprint* se comenzó a trabajar con la posibles versiones de CAD que podría generar la aplicación. También se comenzó a trabajar con el archivo de configuración de usuario, definiendo como seria su gramática e implementando un *parser* para poder validarlo. Para ello se definieron las siguientes tareas:

- Versiones de CAD válidas : Se comprobaría la generación del archivo DXF en diferentes versiones y su validez.
- Definición de gramática para el archivo de configuración de usuario.
- Carga y validación del archivo de usuario.

En este *sprint* se detecta un error en el uso de dos *parsers* que se detalla en la memoria, *Conflicto entre* parsers.

Sprint 9 (04/04/2019 - 10/04/2019)

En este *sprint* se comenzó a trabajar con las sesiones de `Flask` multi-usuario, se continuó con la definición, e implementación de funciones para la gestión del archivo de configuración de usuario y se soluciona el problema de del conflicto entre los dos *parsers*. Para ello se definieron las siguientes tareas:

- Gestión del archivo de configuración de usuario.
- Problema con el uso de dos *parsers*.
- Carga y validación del archivo de usuario.
- Comienzo con las sesiones en `Flask`.

Sprint 10 (11/04/2019 - 24/04/2019)

Este *sprint* se consideró más largo, había una semana de vacaciones en medio. Básicamente se basó en completar las funcionalidades Web, el diseño de la Web , y el tratamiento de los mensajes de aviso. Para ello se definieron las siguientes tareas:

- Añadir funcionalidades a al Web: Añadir o quitar botones, menús, etc.

- Mejorar el diseño de la Web: Página de inicio, barra de navegación superior, página de login, etc.
- Tratamiento de errores: Ver e implementar las distintas formas , en que se presentaran los errores o mensajes de aviso al usuario.

Se solucionaron errores como, el surgido al añadir la letra «ñ» y la tildes, a la gramática del archivo de configuración (se puede ver la solución en la memoria *Codificación UTF-8*). También un error no detectado anteriormente en la función `calculate_azimut_distance`, con un error de división por 0. Además se comprobó que la paleta de colores de los programas CAD, no era la habitual (se puede ver la solución en la memoria *Paleta de colores en CAD*)

Sprint 11 (25/04/2019 - 01/05/2019)

En este *sprint* y como parte final de la aplicación, el objetivo era que la aplicación se desplegara en *Docker*. En un principio integrar la aplicación `Flask` y por último ejecutar todos los contenedores a la vez, con el uso de la herramienta *Docker Compose*. Se obtuvo un resultado correcto.

Sprint 12 (02/05/2019 - 15/05/2019)

En este *sprint* se tratará de redactar la documentación del proyecto.

A.3. Estudio de viabilidad

Viabilidad económica

En este apartado se analizan los costes del proyecto y también los posibles beneficios del proyecto en caso de comercializarse.

Costes

Costes de personal:

El desarrollo del proyecto, tanto el tiempo empleado en la formación, como la implementación de la aplicación y redactar la documentación generada, ha sido llevado a cabo por una sola persona durante 476 horas. Considerando 40 horas semanales, hemos trabajado 2.98 meses, por lo que redondeamos a 3 meses de trabajo a tiempo completo. Si consideramos que el trabajo lo ha realizado un programador junior, el salario será el siguiente:

Concepto	**Coste**
Salario mensual neto	1.088.29 €
Retención IRPF (15 %)	296.,27 €
Seguridad Social (29,9 %)	590,56 €
Salario mensual bruto	1.975,12 €
Total 3 meses	5.925,36 €

Tabla A.1: Costes de personal.

La retribución[2] a la Seguridad Social se ha calculado como un 23,60 % por contingencias comunes, más un 5,50 % por desempleo de tipo general, más un 0,20 % para el Fondo de Garantía Salarial y más un 0,60 % de formación profesional. En total un 29,9 % que se aplica al salario bruto.

El porcentaje de IRPF[7] se ha establecido en el 15 % ya que es el considerado como rendimientos de trabajo para la elaboración de obras científicas.

Costes de *hardware* y *software*:

A continuación se presentan los costes por el emphhardware y *software*, utilizado, en el desarrollo del proyecto. Considerando una amortización a 5 años y se ha usado 3 meses.

Concepto	**Coste**	**Coste amortizado**
Ordenador personal	1.200 €	60 €
Windows 10 Home	145 €	7,25 €
Total	1.345 €	67,25 €

Tabla A.2: Costes de *hardware* y *software*.

Costes de varios:

El resto de costes del proyecto son:

Concepto	**Coste**
Alquiler de oficina	250 €
Internet	120 €
Total	370 €

Tabla A.3: Costes varios.

Costes totales:

El coste total del proyecto es:

Concepto	**Coste**
Personal	5.925,36 €
Hardware & Software	67,25 €
Varios	370 €
Total	6.362,61 €

Tabla A.4: Costes totales.

Beneficios

Se está pensando en distribuir la aplicación comercialmente en un futuro. Un modelo de negocio que se está teniendo en cuenta es, establecer diferentes cuotas por tramos, en función del tamaño de los archivos o del número de puntos a convertir.

Por ejemplo:

Número de puntos	**Cuota**
10.000	10 €/mes
30.000	20 €/mes
Sin Límite	40 €/mes

Tabla A.5: Tipos de suscripciones y cuotas

Añadiendo nuevas funcionalidades, se estudiarán otros parámetros en el modelo de negocio como por ejemplo, si el usuario desea:

- Imprimir el plano.
- Almacenar en la aplicación una base de datos con símbolos.
- Unir varios archivos.
- Modificar la escala y el formato del plano.
- ...

En este caso la idea sería ofrecer paquetes con diferentes funcionalidades a diferentes precios, por ejemplo:

- Paquete básico: Conversión de ficheros.
- Paquete intermedio: Conversión de ficheros, cambios de escalas y formatos.
- Paquete avanzado: Conversión de ficheros, cambios de escalas y formatos, impresión de archivos.
- Paquete completo: Todas las opciones.

Podría existir una funcionalidad gratuita, que permitiera visualizar el archivo, sin permitir nada más, para atraer posibles clientes.

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

B.2. Objetivos generales

B.3. Catalogo de requisitos

B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

C.3. Diseño procedimental

C.4. Diseño arquitectónico

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

D.3. Manual del programador

D.4. Compilación, instalación y ejecución del proyecto

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Bibliografía

[1] Borbón Alexánder and Mora Walter. *Edición de Textos Científicos LaTEX 2017*. Escuela de Matemática,Instituto Tecnológico de Costa Rica, 2017. ISBN 978-9977662275.

[2] Ministerio de Trabajo Migraciones y Seguridad Social. Régimen general de la seguridad social, 2019. URL http://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537. [Online; Accedido 11-Mayo-2019].

[3] Roger Dudler. git - la guía sencilla, 2018. URL https://rogerdudler.github.io/git-guide/index.es.html. [Online; Accedido 12-Enero-2019].

[4] Python Software Foundation. unittest — unit testing framewor, 2019. URL https://docs.python.org/3/library/unittest.html. [Online; Accedido 22-Marzo-2019].

[5] Holger Krekel et al. pytest, 2017. URL https://docs.pytest.org/en/latest/. [Online; Accedido 22-Marzo-2019].

[6] Diego C Martín. Tutorial de git. manual básico con ejemplos, 2018. URL https://www.diegocmartin.com/tutorial-git/. [Online; Accedido 12-Enero-2019].

[7] Agencia Tributaria. Retenciones e ingresoos a cuenta del irpf en el ejercicio 2019, 2019. URL https://www.agenciatributaria.es/AEAT.internet/Inicio/La_Agencia_Tributaria/Campanas/Retenciones/Cuadro_informativo_tipos_de_retencion_aplicables__2019_.shtml. [Online; Accedido 11-Mayo-2019].