Cómo planificar un TFG de Ingeniería Informática

Guillermo Hernández

2024-07-30

Versión en línea: https://www.dih5.es/planificar-tfgii.html.

1 Modalidades para la adjudicación de tema

En nuestro Grado hay dos modalidades para la adjudicación de temas de TFG: los de propuesta abierta, que se asignan por nota media del estudiante, dentro de una lista cerrada, y los de propuesta coordinada con un profesor. En mi opinión, siempre es preferible lo segundo, pues permite trabajar en el tema que realmente se quiera. En esta guía he intentado recoger los principios que suelo seguir para diseñar propuestas de TFG de Ingeniería Informática, con la expectativa de que sea útil para **definir nuevas propuestas de trabajo**, así como para orientar algunas cuestiones relativas al **alcance ya en su realización**.

Descargo de responsabilidad: cada tutor tiene su propia manera de hacer las cosas. Ponte de acuerdo con tu tutor o posible tutor en todo.

2 Parámetros principales

El punto de partida para plantear una propuesta es su **temática y** sus **aspiraciones de nota**, que condicionan el alcance que tenga que tener en su realización. La propuesta puede recoger solo el mínimo razonable, esto es, el compromiso inexcusable para presentar el TFG, pero convendría pensar realmente, en función de la calificación que se desee obtener, qué se debería realizar.

Si estás totalmente perdido, puedes hablar con algún tutor potencial sobre tus aficiones, conocimientos adquiridos y deseados y aspiraciones de nota para que te ayude a preparar la propuesta.

3 Representación del plan curricular en el TFG

Idealmente en un TFG se debe demostrar la **aplicación de los conocimientos adquiridos en la carrera**. Difícilmente todo tendrá cabida, pero lo que sí se debe cuidar es que si hay algo que se pueda aplicar, se aplique.

Para tener una buena panorámica se puede consultar las **materias** (grupos de asignaturas) en PDF aquí o en versión navegable aquí.

En orden de carga de créditos:

- **Programación**: Seguramente el grueso del proyecto sea la programación en sí. Estaría bien pensar, ya en la realización, en aprovechar el conocimiento de estas asignaturas (por ejemplo, utilizando estructuras o tipos abstractos de datos adecuados).
- **Computadores**: En los proyectos de desarrollo no es tan habitual, pero puede ser relevante en otros cercanos al *hardware*.
- **Matemáticas**: Si aparece algún formalismo matemático, convendría desarrollarlo con rigor, aprovechando la formación recibida.
- **Sistemas Operativos**: Si aparece sincronización de hilos es probable que aporte el contenido de la materia.
- **Redes**: Si hay comunicación en red puede aportar mucho. La parte de "seguridad en sistemas informáticos" suele ser siempre relevante.
- **Ingeniería del Software**: Está por todas partes en la estructura del trabajo.
- **Bases de Datos**: Siempre que sea posible (que no sea forzado) se debería introducir un diseño de base de datos en el software.
- **Interacción Persona Ordenador**: Las interfaces deberían diseñarse, evaluarse y mejorarse con los principios de esta materia.
- **Sistemas Inteligentes**: Tienen cabida si se incluye algo de IA.
- **Legislación y Empresa**: Si se trata algún tema legal o más cercano al mundo de la empresa, debe hacerse con el rigor que da la formación en esta materia.
- **Informática Industrial**: Aparecerá en trabajos específicos.

4 Estrategias para aumentar el valor del trabajo

A continuación se recogen algunas ideas para **aumentar el valor** del trabajo realizado. No tienen por qué figurar en la propuesta, pero está bien tener en mente las posibilidades.

- Incluir una **gestión de usuarios**. Casi siempre aporta, y habitualmente los *frameworks* dan mucho trabajo hecho.
- **Empaquetado**: merece la pena trabajar para que el despliegue sea lo más sencillo posible, por ejemplo, creando paquetes en el marco que proporcionen los lenguajes usados.

- **Contenedorización**: Un paso más allá del anterior, se pueden crear contenedores para automatizar al máximo el despliegue.
- **Despliegue en** tecnologías en **la nube**: Se puede usar plataformas como AWS, Azure, Google Cloud (enumeradas alfabéticamente, sin pretensión de prioridad en ello). Tienen un coste, así que se puede acometer en la parte final del desarrollo. Puede haber también soluciones más específicas según tecnologías usadas, como PythonAnywhere.
- **Pruebas unitarias**: Que realmente sean pruebas unitarias, esto es, que estén programadas y se ejecuten de manera automática. Normalmente se realizan con un framework escogido en función del lenguaje empleado.
- **Control de versiones**: Lo más habitual es con Git y GitHub, pero procurando hacer un uso correcto de los *commits* y el sistema de ramas, esto es, investigando bien cómo hacer un uso correcto.

5 Redacción de la propuesta

- Comenzar por poner de manifiesto una carencia o **problema**, presentando después la **propuesta como** una **solución** ("hundimiento y rescate").
- La propuesta tiene que recoger unos **requisitos mínimos** para que el resultado sea aceptable. No hace falta poner nada de más. Es mejor ser ambicioso en la realización que en la propuesta.
- Por si hay que pivotar en el desarrollo, en lugar de fórmulas cerradas como "se usará Python con el *framework* Flask", utilizar **expresiones abiertas** como "se utilizarán lenguajes de programación adecuados al desarrollo, como puede ser Python, posiblemente con algún *framework* como Flask, Django o FastAPI". De hecho, la **elección motivada** de una u otra tecnología podría ser una parte relevante del trabajo, que acabaría reflejada en la memoria.