Requisitos de Buenas Prácticas para el Proyecto

A fin de seguir una serie de buenas prácticas en el desarrollo del proyecto se exponen a continuación una serie de requisitos de **obligado cumplimiento**.

Estructura del Proyecto

El proyecto utilizará obligatoriamente Git para el control de versiones y la colaboración entre miembros del grupo.

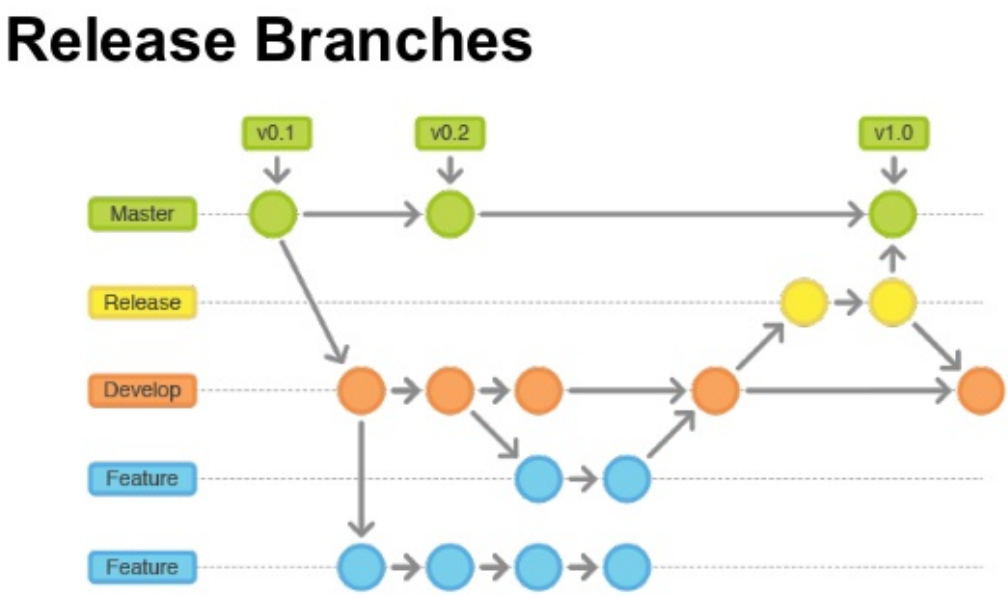
- Cada programa del proyecto (Arduino, Android y Backend) tendrá su propio repositorio en git. NO deben mezclarse los proyectos en un mismo repositorio.
- Cada repositorio tendrá un fichero README con una breve descripción de la aplicación y la forma de ponerla en funcionamiento.
- La documentación de cada aplicación estará en la raíz del repositorio en la carpeta `doc`.
- Los tests de la aplicación estarán en la carpeta `test`.
- El fichero README incluirá instrucciones sobre como compilar la documentación (si es necesario) y como ejecutar los tests.

Uso de Gitflow

Siguiendo la metodología Gitflow para el uso de ramas el proyecto tendrá al menos las siguientes ramas:

- Rama **master** (o **main**): Donde se colocan y etiquetan releases definitivas de la aplicación, ¡nunca se desarrolla sobre master!
- Rama **develop**: Donde se mantiene la rama actual de desarrollo y se añaden nuevas características. Cuando se quiere sacar una nueva release se mergea develop sobre master y se etiqueta.
- Rama **feature/nombrefeature**: Para desarrollar características diferentes en paralelo se pueden crear ramas de tipo *feature* con el nombre `feature/nombredefeature` (ej. `feature/add-testing-module`). Una vez finalizada la feature se mergea sobre develop.
- Rama **release**: Si se considera apropiado se puede usar una rama de tipo release para

preparar las releases antes de llevarlas a master (solo se arreglan bugs, no se añade funcionalidad). La nomenclatura es de tipo `release/numerorelease` (ej. `release/v0.2.3`).



Tablero de trabajo para los sprints

El tablero de trabajo tendrá las siguientes columnas:

- **Product Backlog**: con todas las historias del proyecto
- **Sprint Backlog**: con las historias seleccionadas para el sprint actual
- **Pendiente**: con las historias seleccionadas del Sprint Backlog, que han sido analizadas y desglosadas si es necesario en tareas más pequeñas (es recomendable que todas las tareas tengan una duración similar).
- **Realizándose**: con las historias que están en proceso de desarrollo actualmente.
- **Verificándose**: en esta fase se verifica que la tarea supera las pruebas de aceptación, si no es así no puede pasar a la siguiente columna.
- **Terminado**: las tareas terminadas y verificadas se colocan en esta columna.

Las fichas para las columnas 3,4 y 5 deben tener la siguiente estructura:

Nombre del Campo	*Descripción del Campo*
Título	Título de la tarea. Si se ha desglosado de otra tarea debe reverenciarse (Ej: de la Tarea A puede desglosarse la A1 y la A2)
Responsable	Encargado de la tarea.
Descripción	Descripción detallada de la tarea.
Diseño	Documento de diseño de la tarea. Puede adjuntarse o preferiblemente poner la URL al documento de diseño.
Prueba de Aceptación	Descripción de las pruebas que ha de superar la tarea para ser aceptada. Es el requisito para mover una tarea a la columna Terminado.

Estilo de comentarios

Los comentarios deberán seguir el estándar de generadores de documentación como javadoc, doxygen, sphinx (según lenguaje). Incluirán:

- Una breve descripción del cometido de la función o clase junto con el diseño lógico escrito siguiendo la sintaxis de la asignatura de Programación.
- Los parámetros de entrada (si es un método) con el tipo de datos descrito.
- Lo que devuelve el método con el tipo de datos descrito.
- Si puede lanzar alguna excepción.
- Cualquier otra información que se considere relevante.

Ejemplos en varios lenguajes:

```
var my_function = $(function(){
  /**
   * La descripción de my_function. Usar la sintaxis del documento de di
   *
   * @param {String} param1 - Descripcion de param1.
   * @param {Number} param2 - Descripcion de param2.
   *
   * @returns {Number} Descripcion del valor devuelto.
   */
```

```
/**
 * La descripción de my_function. Usar la sintaxis del documento de dis
 *
 * @param param1 Descripcion de param1.
 * @param param2 Descripcion de param2.
 *
 * @returns Descripcion del valor devuelto.
 */
public int my_function(String param1, int param2) {
```

```
/**
 * La descripción de my_function. Usar la sintaxis del documento de d
 *
 * @param param1 Descripcion de param1.
 * @param param2 Descripcion de param2.
 *
 * @return Descripcion del valor devuelto.
 */
int my_function(string param1, int param2) {
```

```
def my_function(param1, param2):
"""
La descripción de my_function. Usar la sintaxis del documento de dis

Args:
    param1 (str): Descripcion de param1.
    param2 (int): Descripcion de param2.

Returns:
    int: Descripcion del valor devuelto.
"""
```