Sesión 5. Buenas Prácticas en el Desarrollo de Proyectos II.

Bug tracking. Generadores de documentación.

Hipólito Guzmán Miranda
Departamento de Ingeniería Electrónica
Universidad de Sevilla
hipolito@gie.esi.us.es

Parte 1: Bug tracking.

¿Qué es el bug tracking?

Un sistema de bug tracking o seguimiento de errores es una aplicación software que permite anotar y seguir la evolución de fallos o bugs en sistemas en desarrollo, ya sean software o hardware.

El uso de un sistema de seguimiento de errores se considera una buena práctica de diseño no sólo en proyectos de software, donde su uso está más extendido, sino también en proyectos de diseño de hardware, diseño de PCBs o proyectos de sistemas empotrados en los que han de desarrollarse tanto hardware como software. Es una forma de sistematizar la captura y el procesado de los bugs y problemas encontrados en el diseño, ya que en el día a día del trabajo es normal que se olviden unos bugs mientras se están arreglando otros, o desarrollando funcionalidades nuevas.

En esta práctica vamos a trabajar con la herramienta `mantis`, aunque existen otras herramientas para bug tracking como son `Bugzilla`, `Trac` o `FogBugz`. Se pueden encontrar herramientas de distinta complejidad y funcionalidades adicionales, como por ejemplo separación entre bugs y funcionalidades por añadir, integración con sistemas de control de versiones o gestión de proyectos. Según las características del equipo y del proyecto será aconsejable utilizar una herramienta u otra.

Anatomía de un bug:

Un bug report debe incluir siempre, independientemente de la herramienta utilizada:

1. Pasos para reproducirlo
2. Qué esperabas que ocurriera
3. Qué ocurrió realmente

Estos tres puntos son críticos: si falta alguno de ellos, nadie que lea el informe será capaz (por regla general) de reproducir el error, ni mucho menos arreglarlo.

Vida de un bug:

Un bug pasa por una serie de estados desde que es 'reportado' por primera vez hasta que es finalmente marcado como resuelto. Mantis nos ofrece por defecto los siguientes estados:

- New (nuevo) - El estado por defecto para nuevos items.
- Acknowledged (admitido) - Se acepta el informe de bug pero aún no ha sido reproducido por el equipo de desarrollo.
- Confirmed (confirmado) - Se ha podido reproducir el bug.

- Assigned (asignado) - Hay algún miembro del grupo de desarrolladores encargado de resolver el bug (o de añadir la nueva funcionalidad).
- Resolved (resuelto) - El bug ha sido resuelto.
- Closed (cerrado) - El arreglo del bug ha sido probado (idealmente, por una persona diferente a quien lo resolvió, por ejemplo quien informó originalmente del bug).

Los equipos de trabajo pueden poner reglas internas a la evolución de los bugs, y muchos bug trackers pueden configurarse para forzar su cumplimiento. Por ejemplo, que sólo puedan abrir bugs los testers, y que sólo pueda cerrar un bug quien lo ha abierto. En equipos pequeños donde no hay testers dedicados, normalmente no tenemos ese lujo, pero cuando los equipos crecen (y sobre todo si se aceptan incidencias directamente de un cliente) configurar estas reglas puede ser interesante.

Manejo del bug tracker Mantis:

Podéis acceder a mantis en la dirección web `woden.us.es/mantis`

Vamos a informar de un bug dentro del proyecto que tenemos asignado. Podemos llamarlo, por ejemplo, <nombredeusuario>_testing. Primero lo crearemos nuevo, luego nos lo asignaremos a nosotros mismos. Lo iremos pasando de un estado a otro, hasta que lo marquemos como resuelto y si queremos podemos probar la sección de “notas”. Antes de marcarlo como cerrado es interesante que veamos el historial del bug (abajo, en “Issue history”).

Finalmente, uno de los miembros del grupo de trabajo debe añadir un issue a mantis que indique que el core que se añadió en el trabajo no presencial de la práctica anterior no está correctamente documentado (por ejemplo, podéis llamarlo “Core <corename> missing doxygen comments”).

Parte 2: Generadores de documentación.

¿Qué es un generador de documentación?

Un generador de documentación es una herramienta que genera documentación técnica automáticamente a partir de código fuente especialmente comentado (es decir, manteniendo cierta sintaxis en los comentarios, de manera que la herramienta los pueda entender).

En esta práctica vamos a trabajar con la herramienta `doxygen`, aunque por supuesto existen otros generadores de documentación, y dependiendo del lenguaje utilizado habrá que seleccionar uno u otro para nuestro proyecto.

Doxygen soporta los siguientes lenguajes: C, C++, C#, Objective-C, IDL, Java, VHDL, PHP, Python, Tcl, Fortran, D.

Creando un fichero de configuración

El comando `doxygen -g` genera un fichero llamado Doxyfile donde configuraremos cómo se comportará la herramienta en nuestro proyecto.

Doxyfile es auto-descriptivo, porque se crea ya comentado. Vamos a cambiar en el fichero las siguientes opciones:

```
PROJECT_NAME           = “Hola, Mundo”

OUTPUT_DIRECTORY       = doc

OPTIMIZE_OUTPUT_FOR_C  = YES

INPUT                  = src          #Directorio en el que están los fuentes comentados

DISABLE_INDEX          = YES          #Desactiva el índice superior

GENERATE_TREEVIEW      = YES          #Activa la visión en árbol a la izquierda
```

Se recuerda que existe una opción, OPTIMIZE_OUTPUT_VHDL, que habría que habilitar en caso de documentar proyectos VHDL.

También existen más opciones, como GENERATE_MAN, que permite generar páginas de `man` o EXTRACT_ALL = YES, que permite generar documentación de código fuente que no tenga comentarios de doxygen (útil para analizar un código que te hayan pasado). Se recomienda echar un vistazo al resto de opciones y a la documentación.

Sintaxis de doxygen

Para que la herramienta pueda procesar correctamente el código fuente tendremos que utilizar cierto formato en los comentarios.

Cada entidad (función, fichero, etc) en el código, se documenta insertando un comentario de bloque justo antes de la entidad. Para cada entidad, podemos añadir una descripción breve y otra detallada. Ambas son opcionales.

Vamos a usar el siguiente formato de bloque, no obstante en la documentación (última referencia de esta práctica) aparecen otros formatos para comentarios de bloque que también se pueden usar:

```
/**
 * ... texto ...
 */
```

Dentro de los comentarios de bloque podemos encontrar los siguientes campos, entre otros:

Opcionalmente, al principio de cada fichero:

```
@file        <nombre>
@author            <nombre>
@date        <fecha>
```

Para dividir nuestro proyecto en módulos y submódulos:

`@addtogroup <etiqueta> [descripción]` (añade la entidad a un grupo, aunque ya exista)
`@defgroup <etiqueta> [descripcion]` (crea un grupo y añade la entidad a ese grupo, dando error si el grupo ya existe)

Los grupos se pueden anidar unos dentro de otros, como veremos más adelante en un ejemplo.

Para descripciones breves y extendidas:

`@brief <descripción>` (la descripción extendida se pone justo debajo de la breve, dejando una línea vacía entre ambas. Ambas descripciones son opcionales)

Para funciones:

`@param <nombre> [descripción]` parámetros que recibe
`@return <descripción>` valores que devuelve

Listas globales:

`@bug <texto>` bugs conocidos
`@todo <texto>` tareas por hacer

El campo `@date` tiene poco sentido si estamos usando además control de versiones, y los campos `@bug` y `@todo` igualmente son redundantes si gestionamos esas tareas con otro sistema, por ejemplo con un bug tracker.
Si miramos la documentación de la herramienta podemos encontrar campos adicionales.

Hola, mundo!

Vamos a documentar el siguiente ejemplo con doxygen, para poner en contexto la sintaxis.

```
/**
 * @file        main.c
 * @author      Hipólito Guzmán
 * @date        26 Nov 2013
 */

/**
 * @addtogroup common
 * @{
 * @defgroup common_helloworld Hola Mundo
 * @{
 * @brief Prints "Hola, mundo" in the console
 *
 * Extended description goes here
 */

#include <stdio.h>

/**
 * @param void Receives no parameter
 * @return 0, everytime
 * @todo Translate printed message to English
 * @bug Text appears in Spanish, should appear in English
 * @brief main() function of the program
 */
int main (void)
{
  printf("Hola, mundo!!\n");

  return 0;
}

/*
 * @}
 * @}
 */
```

Tendremos que indicarle a doxygen en el Doxyfile dónde puede encontrar este fichero. Lo ideal es meterlo en una carpeta nueva, `helloworld` (o mejor `helloworld/src`), y crear el Doxyfile en `helloworld`.

Ejecutamos doxygen con:

```
doxygen Doxyfile
```

Finalmente, copiamos la documentación html generada a una carpeta desde la que podrá ser vista públicamente usando un navegador web:

```
cp -R doc/html/ /var/www/html/users/<nombreusuario>/helloworld/
```

Accedemos con el navegador a `woden.us.es/users/<nombreusuario>` y comprobamos la documentación html generada.

Documentando el proyecto de la sesión anterior

Documentad el proyecto software que hicísteis durante la práctica anterior y ponedlo en `/var/www/html/groups/pseNN` , de forma que la documentación sea visible en la dirección web `woden.us.es/groups/pseNN`

Trabajo no presencial:

1) Documentando VHDL:

Leed en la documentación de Doxygen la sintaxis de comentarios Doxygen para VHDL

Si el core que añadísteis al repositorio en la sesión anterior está descrito en VHDL:

- Documentad con formato doxygen tres ficheros del core de código libre que añadísteis al repositorio en la sesión anterior. Documenta únicamente la sección ENTITY de cada fichero. Uno de estos ficheros tiene que ser el top-level del core.

Si el core que añadísteis al repositorio en la sesión anterior está descrito en Verilog u otro lenguaje distinto de VHDL:

- Crea un wrapper en VHDL para el top-level del core y documenta con doxygen el wrapper que has creado. Documenta únicamente la sección ENTITY.

Cuando termines, marca como resuelta la incidencia “Core <corename> missing doxygen comments” en el bug tracker.

2) Otras herramientas:
Busca un generador de documentación (no importa si es gratis/de pago o libre/propietario) que soporte al menos un lenguaje no soportado por Doxygen (como estamos en la especialidad de electrónica, puede ser interesante buscar alguno que soporte Verilog/SystemVerilog, AHDL -de Cadence-, o incluso UML)

El trabajo no presencial se evaluará durante las sesiones de seguimiento.

Agradecimientos:

A Luis Sanz por introducir estas herramientas en nuestro grupo de trabajo y por su inestimable ayuda para preparar esta práctica.

Referencias:

Mantis Bug Tracker, http://www.mantisbt.org/
Doxygen documentation: www.doxygen.org
Doxygen documentation, “Comment blocks for C-like languages”: http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html