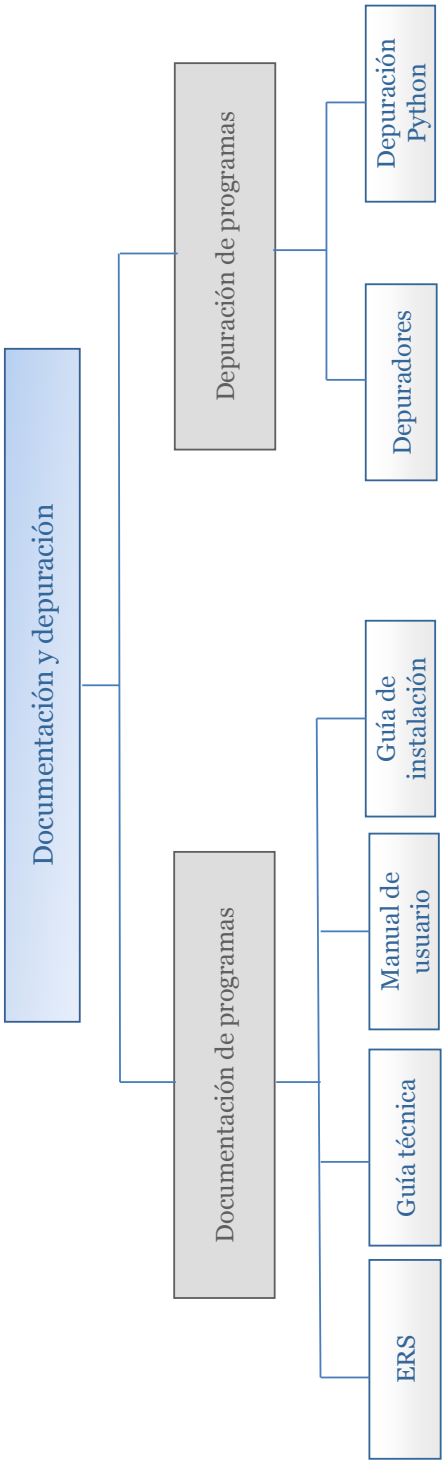
Documentación y depuración

TEMA 10

Esquema


Documentación y depuración
Depuración de programas
Depuración Python
Depuradores
Documentación de programas
Guía de instalación
Manual de usuario
Guía técnica
ERS

Ideas clave

10.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás comprender las **Ideas clave** expuestas en este documento, que se complementan con lecturas y otros recursos para que puedas ampliar los conocimientos sobre el mismo.

En este tema se explica en qué consisten la documentación y depuración de programas. Por tanto, el alumno debe prestar especial atención a:

- Documentación de **especificaciones**
- **Guía técnica**
- **Manual de usuario**
- **Guía de instalación**
- **Depuración** de programas
- **Depuración** de programas en **Python**

10.2. Documentación de programas

La **documentación de programas** es una parte esencial en el desarrollo de aplicaciones informáticas, ya que permite realizar un **mantenimiento** de la misma, **control de calidad** o un **uso adecuado** del producto, así como **reutilización del código** en otros programas.

Como se vio en el Tema 2, el desarrollo de un producto de software se realiza en distintas etapas: análisis, diseño, codificación, verificación y validación, verificación y prueba, explotación y mantenimiento.

No se debe esperar a las etapas finales del ciclo de vida para iniciar la documentación de programas, sino que cada una de las etapas deben ir documentándose. De esta manera, en el paso de una etapa a otra, resultará mucho más claro qué tareas deben realizarse.

Por ejemplo, si se empezara a escribir código sin tener en cuenta la documentación generada en las fases de análisis y diseño, se producirá un software mal diseñado que no cumplirá con todos los requisitos. Los costes asociados a la depuración de los errores surgidos serán muy elevados.

También es muy importante que la documentación esté actualizada y se corresponda con la última versión de los programas, por tanto, tiene existir documentación relativa a las configuraciones de cada versión del programa. Evidentemente, la versión que se escoja, tendrá que ser la que se ha instalado y sobre la que se han realizado las pruebas correspondientes.

10.3. Documentación de especificaciones

La **especificación de requisitos de software** (ERS) es un documento que debe describir qué tiene que hacer el producto.

Esta parte es esencial para poder diseñar el producto de manera adecuada y para asegurarse de que el cliente está de acuerdo en las funciones y requisitos del producto (Figura 12.1).

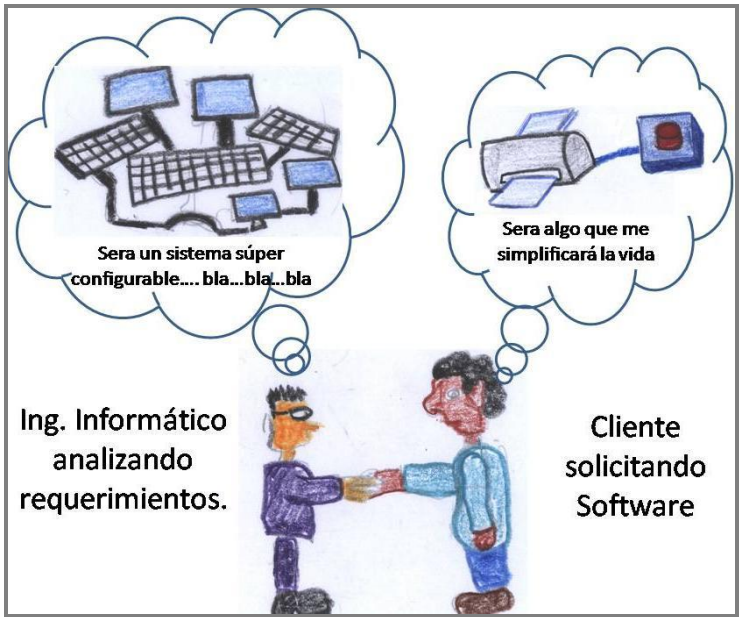


Figura 12.1: Representación de los puntos de vista del cliente y el desarrollador en la ERS.
Fuente: http://solucionatuproblemacoco.blogspot.com.es/

Según el estándar de IEEE (Institute of Electrical and Electronic Engineers), IEEE 830, que suele tomarse como referencia, el documento de especificaciones debe contener (IEEE (1984)):

Introducción

- **Propósito.** Se especifica cuál es el propósito del documento y a quién va dirigido.
- **Ámbito del Sistema.** Se da nombre al producto, se explica qué hará…
- **Definiciones, Acrónimos y Abreviaturas.** Definir todos los términos, acrónimos y abreviaturas del texto**.**
- **Referencias.** De todas las referencias del documento, hay que prestar especial atención a los relacionados con los requisitos.
- **Visión General del Documento.** Descripción del contenido y organización del documento.

Descripción General

Se explican los factores que afectan al producto y a sus requisitos, es decir, su contexto. De esta manera, cuando se aborden los requisitos en la siguiente sección, serán más fáciles de entender.

- **Perspectiva del Producto.** Si el software que se va a diseñar tiene relación con otro software (por ejemplo, forma parte de un proyecto mayor), hay que especificarlo en este subapartado.

- **Funciones del Producto.** Se describe de manera general qué va a hacer el producto.

- **Características de los Usuarios.** A la hora de desarrollar un producto es imprescindible tener en cuenta cuál es el perfil del usuario final (formación académica, experiencia técnica, etc.) y debe exponerse en este subapartado.

- **Restricciones.** Las limitaciones impuestas a los desarrolladores a la hora de desarrollar el producto, como limitaciones del hardware, lenguaje de programación, consideraciones acerca de la seguridad, etc., se especifican en esta subsección.

- **Suposiciones y Dependencias.** Describe aquellos factores de los que dependen los requisitos del programa, como puede ser el sistema operativo, el organigrama de la empresa, etc. Así, si estos factores cambian deben revisarse los requisitos para evitar sorpresas desagradables.

- **Requisitos Futuros.** Se especifican las mejoras futuras del producto.

Requisitos Específicos

Esta es la sección más larga e importante del documento. Se especifican los requisitos con un nivel de detalle que permita diseñar un producto de software que sea eficaz, así como planificar las pruebas correspondientes. En su redacción, debe tenerse en cuenta que:

- El contenido del documento debe ser perfectamente entendible por personas que no tengan un perfil técnico.
- Todos los requisitos deben estar identificados con un código único.
- Los requisitos tienen que reflejar una necesidad real.
- No debe haber ambigüedad en la interpretación de los requisitos.
- Todos los requisitos relevantes tienen que estar incluidos.
- Los requisitos no pueden ser contradictorios.
- Hay que clasificar los requisitos (generalmente, en función de su importancia).
- Hay que contemplar posibles cambios en los requisitos y tratar de que puedan hacerse de la forma más estandarizada y controlada posible (Figura 12.2).
- Los requisitos deben ser trazables, esto es, hay que saber su origen y su relación con el diseño del producto.

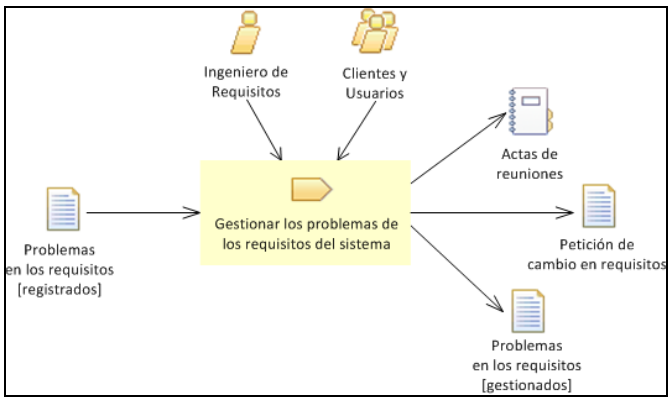


Figura 12.2: Gestión de cambios y problemas en la especificación de requisitos.
Fuente: http://www.juntadeandalucia.es/

- **Interfaces Externas.** En este subapartado se describen los requisitos de las interfaces (de usuari, con otros sistemas, etc).

- **Funciones.** En esta sección se explican todas las funciones, es decir, todas aquellas tareas que debe llevar a cabo el producto. Conviene tenerlas clasificadas por objetivos, por usuarios, etc.

- **Requisitos de Rendimiento.** Son los requisitos relacionados con la carga esperada. Por ejemplo, el número de usuarios conectados simultáneamente o es número de transacciones por segundo.

- **Atributos del Sistema.** Se detallarán cuestiones como la seguridad o portabilidad del sistema.

- **Otros Requisitos.** Se incluirá cualquier otro requisito que se quiera incluir y que no pertenezca a ninguna de las categorías anteriores.

Apéndices

En este apartado se puede incluir información relacionada con el proyecto que no haya sido incluida previamente, como los formatos de datos o los costes del proyecto.

10.4. Guía técnica

En la guía técnica se especifican tres aspectos esenciales de un producto de software: el diseño de la aplicación, el código fuente y las pruebas que se han realizado. Es un documento que, a diferencia de las especificaciones, está hecho para el personal técnico (Herranz & Quero, 2000). De hecho, tiene como objeto servir de guía para el desarrollo del software, lo que incluye la codificación y la corrección de errores, así como facilitar las futuras actualizaciones.

Está formada por varios documentos, que se agrupan atendiendo a los aspectos principales: cuaderno de carga, programa fuente, pruebas.

Cuaderno de carga

El cuaderno de carga contiene las especificaciones técnicas, basándose en la documentación de especificaciones. En otras palabras, a partir de las necesidades del cliente (recogidas en la documentación de especificaciones) especifica a los programadores cómo deben desarrollar el código.

Con este documento se busca que el desarrollo sea más rápido y se ajuste a los requisitos que el cliente ha pedido. También se emplea para saber cómo dividir el trabajo entre varios programadores.

En el cuaderno de carga se suelen considerar las secciones

- **Tratamiento general**. En esta sección se describe de forma general cómo se va a plantear el código, ajustándose a las características del sistema en que se va a implantar. En este apartado se incluyen:
  - **Descripción general de la aplicación**. Describe las tareas que se van a llevar a cabo para tener una visión de conjunto.
  - **Especificaciones del sistema.** Se especifica todo lo relacionado con la arquitectura del sistema, esto es, tipos de ordenadores, sistema operativo, tipo de red, bases de datos, etc.
  - **Plan de trabajo**. Consiste en planificar la ejecución de las tareas, atendiendo a los recursos disponibles, tratando de minimizar costes.

- **Diseño de datos**. Consiste en describir los ficheros de datos que se van a utilizar durante la aplicación. Es necesario especificar:
  - **Relación de ficheros**. Deben especificarse y describirse todos los tipos de fichero con que va a trabajar el programa y asignar un código a cada tipo.
  - **Descripción de fichero**. Para cada fichero, hay que especificar cuestiones como su formato, el número máximo de registros, la frecuencia de utilización, el programa de creación, etc.
  - **Descripción de registro**. Debe especificarse cómo serán los registros asociados a cada uno de los ficheros. Debe incluirse información como características de cada campo, longitud del registro en bytes, campo clave, etc.

- **Diseño de entrada-salida**. En esta etapa se especifica el diseño de los formularios y las pantallas del programa. El diseño se especifica de forma gráfica, apoyado texto con las descripciones que pudieran hacer falta.

- **Diseño modular**. En esta etapa se define la estructura del producto y se describen sus módulos principales. En esta etapa se describen:

  Diagrama general. Es conveniente tener un esquema de los principales módulos del programa (Figura 12.3)
  Descripción de módulos. Se detalla cada módulo del diagrama general, especificando su función.

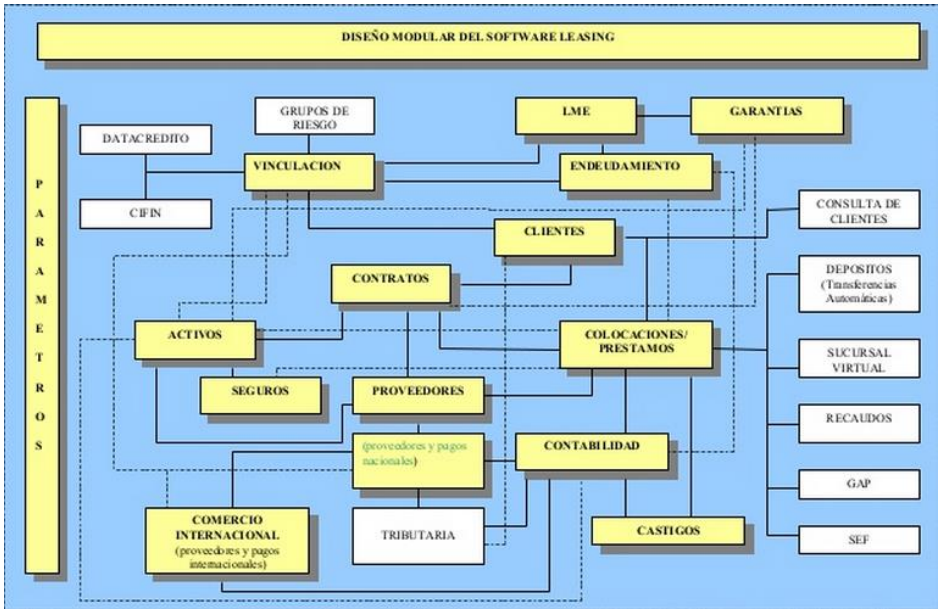


Figura 12.3: Diseño modular del software de leasing.
Fuente: http://es.slideshare.net/martiniturbide/

- **Diseño de programas**. Hay que describir detalladamente los programas en que se divide el producto de software con el fin de facilitar las pruebas, el mantenimiento o el reparto del trabajo. Fundamentalmente, hay que describir el nombre y la descripción de tareas que realiza cada programa, las variables que utiliza cada uno de ellos y su ámbito, etc. También hay que detallar el algoritmo del programa, utilizando pseudocódigo u organigramas (Figura 12.4).

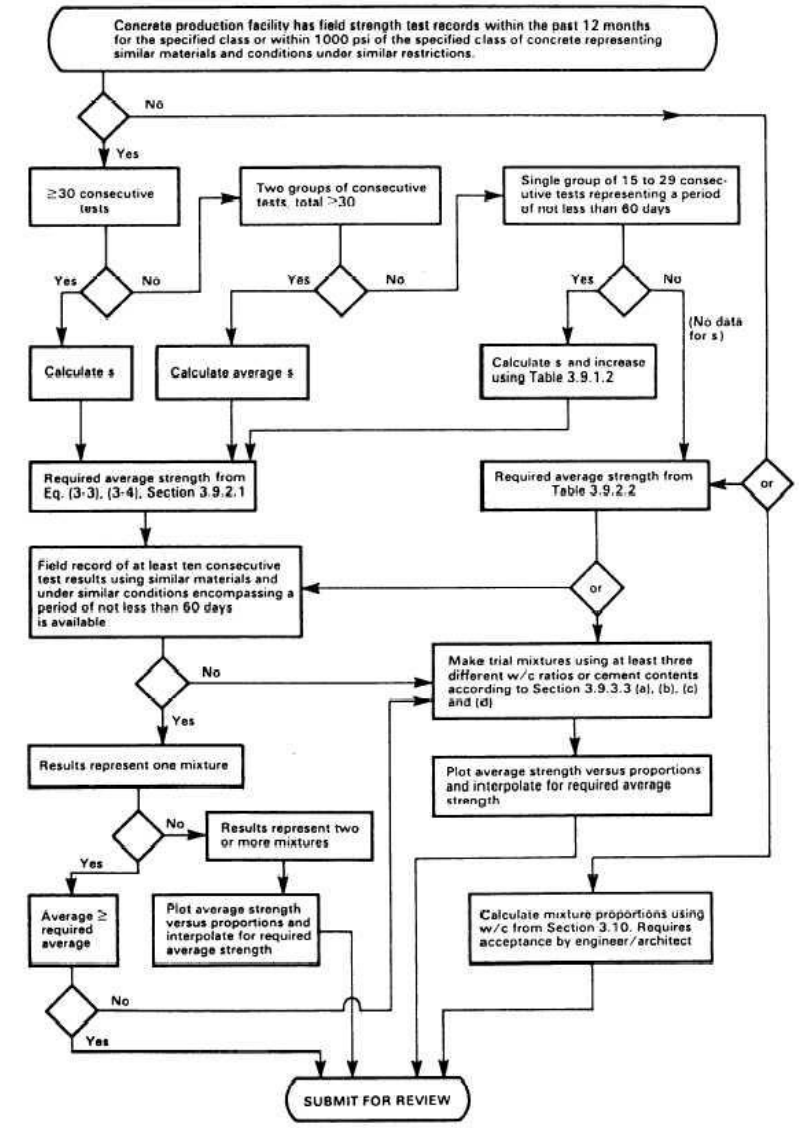


Figura 12.4: Organigrama de un programa para documentación.
Fuente: http://draftingmanuals.tpub.com/

Con toda esta documentación preparada, ya se puede empezar a desarrollar el software.

Programa fuente

El **programa fuente** es el documento en el que se escribe el **desarrollo de todos los programas** del producto de software.

Es importante que el código esté **comentado**, ya que debe ser entendible para realizar mantenimientos futuros por desarrolladores que, en muchos casos, no van a haber participado en la codificación del programa. Por la misma razón es importante que el código esté correctamente **indentado** y su **sintaxis** sea lo más **clara** posible.

Pruebas

Una vez terminado el desarrollo del programa, se entra en la etapa de verificación y validación, que consiste en la realización de pruebas para confirmar que el programa funciona según lo previsto o para detectar fallos que deben ser corregidos.

La realización de pruebas también debe ser documentada, indicando además si se tratan de pruebas de unidad (se prueba cada módulo por separado), de integración (se prueban distintos módulos combinados entre sí) o de sistema (se prueba el funcionamiento todo el sistema).

10.5. Manual de usuario

El **manual de usuario** es un guía para que los distintos tipos de usuarios puedan utilizar el programa correctamente.

La información de este manual se extrae de la guía técnica, pero debe tenerse en cuenta que los usuarios, por lo general, no van a tener un perfil técnico, cosa que debe tenerse en cuenta en la elaboración del material.

Por tanto, la guía de usuario, **debe centrarse en la entrada y salida de la información** con que trabaja el programa, dejando de lado la parte técnica y debe estar redactada con claridad y sencillez. Debe incluirse también un glosario de acrónimos y términos técnicos.

Al contrario de lo que pueda parecer, puede ser conveniente redactar el manual antes de desarrollar el programa. De esta forma, si hubiera algún error u omisión en la entrada o salida de datos, se puede detectar fácilmente revisando el manual.

Los manuales de usuario suelen incluir:

- Índice
- Modo de empleo del manual
- Descripción del programa
- Modo en que se ejecuta el programa
- Descripción de los procesos

- Descripción de los datos de entrada
- Relación de pantallas y descripción de sus formatos
- Descripción de los datos de salida
- Ejemplos de uso
- Solución a problemas habituales y mensajes de error

10.6. Guía de instalación

La **guía de instalación** es el manual que contiene la información para que el producto pueda ser ejecutado.

En primer lugar, es conveniente incluir el modo en que se va a migrar los datos, de la antigua aplicación a la nueva y, una vez hecho el trasvase, las pruebas que hay que realizar mientras los dos sistemas (el nuevo y el antiguo) están funcionando en paralelo. Es necesario depurar al máximo el software nuevo antes de eliminar el antiguo, especialmente si se va a trabajar en producción.

La guía de instalación también debe incluir las normas de ejecución y seguridad

- **Normas de ejecución**. Está relacionadas con:
  - Instrucciones para el técnico del sistema.
  - Descripción de los mensajes del sistema.
  - Procesos que se llevan a cabo, jerarquizados según su prioridad.
  - Administración de los datos que maneja el programa.

- **Normas de seguridad**. Están relacionada con:
  - Seguridad física del sistema. Incluye las normas de protección contra elementos que puedan dañar físicamente el sistema como el agua, las altas temperaturas, etc. así como la política de backups.
  - Seguridad de la información. Son las medidas que deben tomarse para que solo puedan acceder a la información usuarios autorizados. Para ello, hay que definir distintos niveles de usuario con distintos privilegios, además de evitar el acceso de personas ajenas a la empresa.

10.7. Depuración de programas

Cuando se revisan y corrigen los errores del software se realiza una **depuración de programas**.

Es muy frecuente llamar a los errores de programación con el anglicismo ***bug*** y a la depuración *debugging*. A estas alturas del curso, el alumno se habrá enfrentado a varios errores de sintaxis en Python y habrá visto cómo el intérprete imprime un mensaje de error especificando el número de línea donde lo ha encontrado.

En esta situación, los errores no parecen muy difíciles de depurar, basta con leer el mensaje de error y, cuanta más experiencia se tenga, más rápidamente se corregirán. El problema es que los defectos de código no siempre son así de evidentes. Cuando se habla de depuración de programas se hace referencia a la eliminación de los errores que hacen que **el programa no se comporte según lo esperado**.

Por ejemplo, puede ocurrir que un programa arroje un cálculo distinto del esperado o que una pantalla de usuario se bloquee si se rellena con una determinada combinación de campos. También puede ocurrir que los errores solo sean visibles cuando se combinan distintos módulos entre sí (de ahí los distintos tipos de pruebas que hay que realizar) o con determinados tipos de datos. Si estos errores no son detectados y corregidos a tiempo, pueden causar grandes pérdidas a la compañía.

Un ejemplo muy conocido, que incluso salió en la prensa, es el de la sonda Mars Climate Orbiter, que lanzó la NASA en 1998 y que se destruyó por un problema de conversión de unidades: olvidaron transformar los kilómetros a millas.

No debe pensarse que los *bugs* son propios de programadores inexpertos o poco atentos. En la práctica, es imposible producir un código libre de defectos, no solo debido a que a veces los desarrolladores pueden no ver los defectos de su propio código y necesitan otro par de ojos que les ayude, sino también debido a la complejidad de las aplicaciones informáticas, que hace imposible tener una visión detallada de todo el código que se está ejecutando.

El problema es que, por lo general, los errores de código son **muy difíciles de detectar y, sobre todo, de aislar**. Recordemos que los productos de software del mercado suelen ser aplicaciones muy complejas, con gran cantidad de funciones y

posibles combinaciones. Por tanto, que un fallo se detecte en un determinado programa o pantalla no significa que se haya producido allí, puede remontarse muchas líneas atrás o incluso pertenecer a un módulo distinto.

Esto implica que, en muchas ocasiones, la verdadera causa del *bug*, no tiene conexión aparente con su síntoma. Por tanto, resulta esencial no solo saber reproducir el error, sino tratar de reducir al máximo el número de pasos para reproducirlo.

Reportar un bug

Cuando se detecta un fallo en el código, bien como resultado de una prueba, bien por un cliente que está usando la aplicación, se inicia el proceso de reporte del error (Figura 12.5). Esto debe hacerse de forma sistemática y detallada. Además, debe documentarse la solución que se haya dado a cada *bug*.

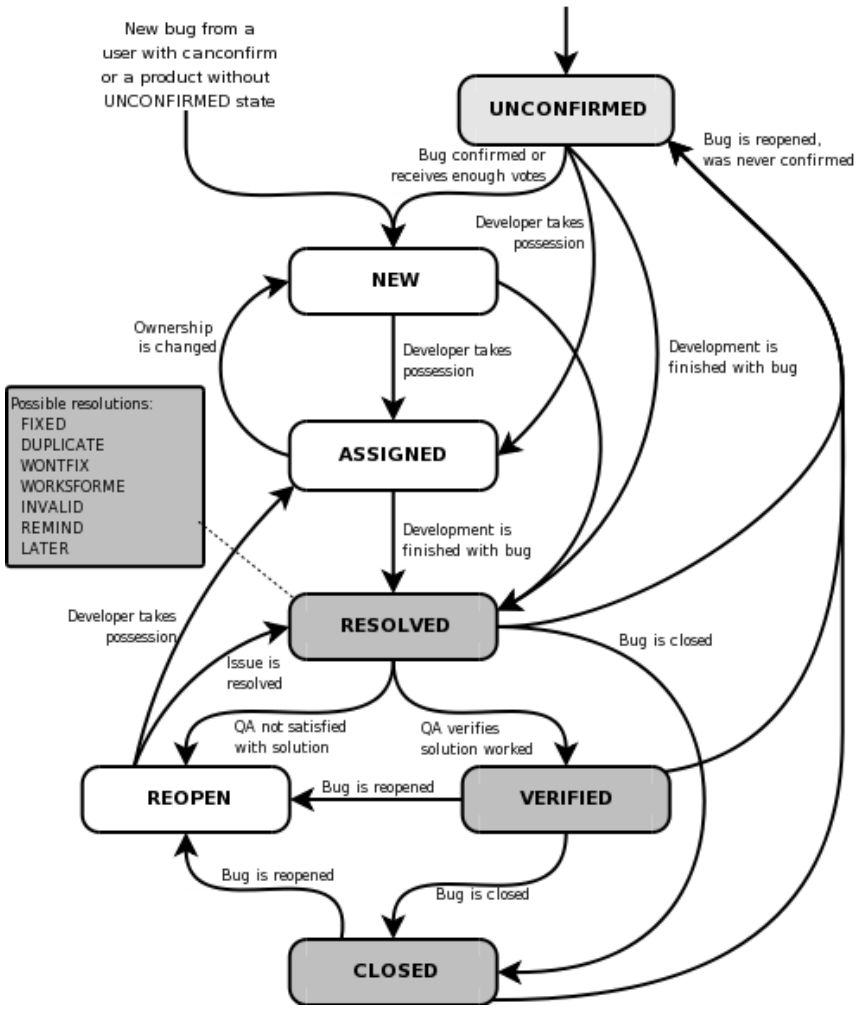


Figura 12.5: Proceso de resolución de un bug. Fuente: http://www.bugtrackingtools.net/

Para programas no muy grandes este seguimiento podría hacerse con un fichero de texto, pero en entornos profesionales y para grandes aplicaciones, lo habitual es tener una base de datos que no solo permitan reportar *bugs*, sino tareas como realizar el seguimiento del mismo, consultar la documentación asociada, etc.

En este tipo de bases de datos se suele incluir información como

- Sistema operativo sobre el que se ejecuta el programa.
- Pasos para reproducirlo.
- Comportamiento obtenido y comportamiento esperado.
- Localización (pantalla o url asociada a error).
- Versión de la aplicación.
- Prioridad.

Figura 12.5: Ejemplo de pantalla para el reporte de un *bug*.
Fuente: https://developer.salesforce.com/

Depuradores

Los **depuradores** son programas que se utilizar para facilitar la tarea de encontrar los errores del código.

Los depuradores permiten **interrumpir la ejecución del código** para comprobar qué está haciendo o verificar el estado de las variables.

Si el lenguaje es compilado, detener la ejecución en cualquier momento no es trivial. Recordemos que los lenguajes compilados traducen todo el código a código máquina y luego se ejecuta. Este proceso no es reversible, es decir, de un lenguaje compilado se puede pasar a un lenguaje máquina, pero cuando se tiene el lenguaje máquina no es posible reconstruir el código original.

Los depuradores permiten superar esta dificultad. No obstante, los lenguajes interpretados como Python son más fácilmente depurables, ya que en estos leguajes el programa se va traduciendo y ejecutando línea a línea.

En la depuración de un programa es esencial el uso de *breakpoints* o **puntos de detención**. Son puntos en los que se detiene la ejecución y se comprueba el estado de las variables. Qué puntos de detención escoger es algo que no puede determinarse de manera teórica y general, solo la práctica puede ir orientado al respecto. Es por estas cuestiones por lo que se suele decir que el depurado de programas es un *arte*.

Depuradores en Python

Existen varias formar de depurar código en Python. Aquí se explica cómo hacerlo con el módulo de Phyton `pdb` (Python DeBugger).
Vamos a ver su funcionamiento con un ejemplo. Supongamos que queremos revisar un fichero que contiene el código

```
a = "primera variable"
b = 1234
c = [1, "a"]
ab = a + str(b)
c.append(ab)
print(c)
```

En primer lugar, hay que importar el módulo `pdb` y en segundo indicar el punto a partir del cual se quiere dejar una traza o rastro (supongamos que es en `b = 1234`). Esto se hace con la función `pdb.set_trace`. Por tanto el código quedaría

```
import pdb

a = "primera variable"
pdb.set_trace()
b = 1234
c = [1, "a"]
ab = a + str(b)
c.append(ab)
print(c)
```

Al ejecutarlo, devuelve lo siguiente

```
-> b = 1234
(Pdb)
```

Es decir, se detiene donde encuentra la sentencia `pdb.set_trace()`, imprime la sentencia actual `b = 1234` (la línea que se ejecutará a continuación) y espera que el usuario introduzca por teclado la siguiente instrucción, que puede ser:

- `p` (print) seguido del nombre de una variable y presionar la tecla ENTER. Es una de las acciones más importantes de `pdb`, ya que devuelve el valor de la variable introducida. Si se ejecuta `p a` devuelve

  ```
  'primera variable'
  (Pdb)
  ```

  También sirve para entender cómo funciona pdb, ejecutando el código línea a línea, ya que se si ejecuta `p b` devuelve

  ```
  *** NameError: name 'b' is not defined
  (Pdb)
  ```

  Si se quiere que devuelva el valor de varias variables, simplemente hay que separarlas por comas.

- `n` (next) y presionar la tecla ENTER. En este caso, se ejecutará la siguiente sentencia. En nuestro ejemplo:

  ```
  -> c = [1, "a"]
  (Pdb)
  ```

- Presionar la tecla ENTER. Si a continuación presionamos ENTER, se mostrará

```
-> ab = a + str(b)
(Pdb)
```

que es lo mismo que se mostraría si hubiéramos presionado `n` y la tecla ENTER. Esto es porque la tecla ENTER vuelve a ejecutar el último comando introducido, en este caso, `n`.

- `q` (quit) y presionar la tecla ENTER. Con esto quita el modo de depuración y vuelve a salir el *prompt*.

- `c` (continue) y presionar la tecla ENTER hace que se termine el *debugging* pero sin salirse del programa, es decir, lo ejecuta hasta el final sin detenerse línea a línea. Esto es muy útil si, por ejemplo, ya se ha visto la parte del código que se deseaba analizar y se quiere ir al final, sin interrumpir la ejecución y sin tener que pulsar `n` + ENTER hasta el final del programa ya que aún pueden quedar muchas líneas por delante que conviertan la tarea en larga y tediosa de manera innecesaria.

- `l` (list) y presionar la tecla ENTER devuelve la parte del programa en que se está. Para un programa tan sencillo como el del ejemplo no sería necesario, en cambio para programas largos es esencial, ya que, al ir ejecutándose línea a línea, es muy fácil perder el punto de vista general. Así, si se ejecuta en nuestro programa devuelve

```
(Pdb) l
  3
  4    a = "primera variable"
  5    pdb.set_trace()
  6    b = 1234
  7    c = [1, "a"]
  8  ->      ab = a + str(b)
  9    c.append(ab)
 10    print(c)
(Pdb)
```

- `s` (step into) y presionar la tecla ENTER se emplea cuando se quiere saltar línea a línea dentro de un subprograma. Esta situación será muy común en programas grandes. Para ver en qué consiste vamos a realizar una modificación del código de ejemplo

```
import pdb

def subrutina(cadena):
    lista = list(a)
    return lista

a = "primera variable"
pdb.set_trace()
b = 1234
c = [1, "a"]
ab = a + str(b)
l = subrutina(ab)
c.append(l)
print(c)
```

Si al ejecutarlo vamos saltando línea a línea (`n` + ENTER), cuando lleguemos a la línea

```
-> l = subrutina(ab)
(Pdb) n
```

Producirá

```
-> c.append(l)
(Pdb)
```

Si, por el contrario, queremos entrar la función subrutina, habrá que ejecutar

```
-> l = subrutina(ab)
(Pdb) s
```

Que produce

```
-> def subrutina(cadena):
(Pdb) n
-> lista = list(a)
(Pdb)
```

- `r` (return) y pulsar la tecla ENTER hace algo similar a `c` + ENTER, pero dentro de una subrutina. Se utiliza para salir de una función que está dentro del programa principal sin detener el programa principal ni salir del modo *debugging*.

Hay más comandos y métodos definidos dentro del módulo `pdb`, aquí solo se muestran algunos de ellos. Para obtener un listado de todos puede ejecutarse la ayuda, `h` + ENTER.

Supongamos que hemos encontrado la línea en la que pensamos que está el error y lo queremos corregir. No es necesario salir, cambiar el fichero y volverlo a ejecutar, puede hacerse todo directamente.

Por ejemplo, supongamos que nos hemos dado cuenta de que nos falta definir, antes de la variable `c`, una variable `k = "La variable que faltaba"`. Podemos proceder así

```
(Pdb) k = "La variable que faltaba"
(Pdb) c
```

que ejecutará el código con la modificación para probar cómo resulta.

En ocasiones puede dar lugar a un error o una notificación extraña. Por ejemplo, supongamos que la nueva variable no queremos que se llame `k` sino `h`. Entonces ejecutaríamos

```
(Pdb) h = "La variable que faltaba"
*** No help for '= "La variable que faltaba"'
```

Lo que ocurre es que se ha confundido el nombre de la variable, `h`, con el comando de ayuda de `pbd`, `h` + ENTER. Para eliminar la confusión debe ejecutarse con el símbolo ! primero:

```
(Pdb) !h = "la variable que faltaba"
```

10.8. Referencias bibliográficas

IEEE (1984) Guide to Software Requirements Specifications IEEE Std 830. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=278253&isnumber=6883

López Herranz, J. & Quero Catalinas, E. (2000). *Fundamentos de programación*. Madrid: Paraninfo.

Lo + recomendado

No dejes de leer…

Un ordenador destruye la sonda «Mars Polar Lander»

Villatoro, R. (s.f.). Un ordenador destruye la sonda «Mars Polar Lander». Recuperado de: http://www.lcc.uma.es/~villa/noticias/marspolar.html

En este enlace se describen algunos errores de programación que han tenido gran impacto y se habla de la imposibilidad de generar un código completamente depurado.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:
http://www.lcc.uma.es/~villa/noticias/marspolar.html

Los peores bugs del hardware

Pardo, L. (15 febrero 2011). Los peores bugs del hardware. Recuperado de: http://www.neoteo.com/los-peores-bugs-de-hardware/

En este enlace se describen errores de hardware especialmente sonados.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:
http://www.neoteo.com/los-peores-bugs-de-hardware/

No dejes de ver...

The software requirements specification

En el siguiente vídeo se muestran los requisitos de software.

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:
https://www.youtube.com/watch?v=_XTQjKhh6hQ

Tips for writing clear requirements

En el siguiente enlace se explican técnicas para escribir requerimientos de forma clara.

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:
https://www.youtube.com/watch?v=IDkyBkUtPLM

+ Información

A fondo

Pdb — The Python Debugger

Se describen las funciones y argumentos del *debugger* de Python.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:
https://docs.python.org/3/library/pdb.html

Seguimiento de errores fácil

Spolsky, J. (16 Abril 2003). Seguimiento de errores fácil. Recuperado de: http://spanish.joelonsoftware.com/Articles/PainlessBugTracking.html

Se describe cómo debe hacerse un seguimiento de errores eficaz.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:
http://spanish.joelonsoftware.com/Articles/PainlessBugTracking.html

Las Cinco Principales Razones (Equivocadas) por las Cuales no tienes Ingenieros de Prueba

Spolsky, J. (30 Abril 2003). Las Cinco Razones (Equivocadas) por las Cuales no tienes Ingenieros de Prueba. Recuperado de: http://spanish.joelonsoftware.com/Articles/TopFiveReasonsYouDontHave.html

Se explica la importancia de la realización pruebas así como de personal competente que las realice.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:
http://spanish.joelonsoftware.com/Articles/TopFiveReasonsYouDontHave.html

Herramientas de documentación ágiles

Talens-Oliag, S. (2006). Herramientas de documentación ágiles. *Revista del Instituto Tecnológico de Informática, 11*.

Se explica la importancia de la documentación y herramientas para generarla de manera más productiva.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:
http://www.iti.es/media/about/docs/tic/11/articulo1.pdf

6 ejemplos en los que un bug se volvió una característica del *software* (misbugs)

Honorio, J. (s.f). 6 ejemplos en los que un bug se volvión una característica del software (misbugs). Recuperado de: http://www.softwero.com/2014/11/6-ejemplos-en-los-que-un-bug-de.html

Se ponen 6 ejemplos de bugs que han pasado a formar parte de las características oficiales de distintos *softwares*.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:
http://www.softwero.com/2014/11/6-ejemplos-en-los-que-un-bug-de.html

Test

1. La documentación de programas:

A. Debe realizar en las etapas finales del ciclo de vida.

B. Debe realizarse después de terminar el código.

C. Debe realizarse en cada etapa.

2. La documentación de especificaciones:

A. Es para personas con un perfil técnico.

B. Debe de poder ser entendible por personas que no sean técnicos.

C. Al ser un documento preliminar, no es muy importante su legibilidad.

3. La redacción de las especificaciones:

A. Emplea metodologías distintas en función de cada proyecto.

B. Suele tomar como referencia el estándar IEEE 830.

C. No es necesaria en todos los proyectos.

4. ¿Qué conjunto de documentos forman la guía técnica?

A. Documentación de especificaciones, cuaderno de carga y manual de usuario.

B. Cuaderno de carga, programa fuente y relación de pruebas.

C. Manual de usuario, guía de instalación y programa fuente.

5. La guía técnica:

A. Es una guía general, redactada para todo tipo de usuarios.

B. Deja de ser útil cuando se completa el desarrollo del software.

C. Sirve de base para elaborar el manual de usuario.

6. El manual de usuario:

A. Debe centrarse en la entrada y salida de datos que maneja el programa.

B. Debe incluir un glosario de términos, acrónimos y abreviaturas.

C. A y B son correctas.

7. Los errores de código:

A. Son exclusivos de programadores inexpertos.

B. Son inevitables en cualquier aplicación.

C. Se solucionan todos cuando la sintaxis es correcta.

8. Detectar y solucionar errores de código:

A. Requiere aplicar una metodología fija en todos los casos.

B. Se considera un arte por su dificultad.

C. No es una operación crítica en el desarrollo de un software.

9. La depuración de código:

A. En general, es más sencilla en lenguajes interpretados.

B. En general, es más sencilla en lenguajes compilados.

C. No tiene ninguna relación con el tipo de lenguaje.

10. El módulo `pdb` de Python:

A. Permite explorar un programa y cualquier función definida en el mismo fichero.

B. Permite al programador ejecutar otras sentencias que modifiquen el resultado de la función.

C. A y B son ciertas.