UNIVERSIDAD DE DEUSTO

Facultad de Informática

Un Modelo Descriptivo de la Forma de Depurar Programas de los Programadores

TESIS DOCTORAL presentada por Cecilio J. Fernández Pérez

DIRIGIDA por el Dr. Anselmo del Moral Bueno

El Director

El Doctorando

Mayo, 1991

MI AGRADECIMIENTO

Al Director de esta Tesis, Dr. Anselmo del Moral.

A todos los profesores de mi departamento.

A todos los estudiantes que han intervenido en las diversas pruebas.

Estoy en deuda con todos ellos, ya que sin su ayuda no habría sido posible realizar esta Tesis.

A Isabel,

RESUMEN

Esta tesis presenta los resultados obtenidos de tres experimentos, efectuados en un medio académico, sobre el proceso de depuración de programas, seguido tanto por programadores expertos como inexpertos.

Asi mismo, propone un modelo de comportamiento en la depuración de programas, para distintos grupos de programadores, concretamente de nula experiencia (novatos), de experiencia media (intermedios) y finalmente de mucha experiencia (expertos).

LABURPEN

Tesi honek, hezkuntzako inguruncan oinarriturik, programen arazketako prosezoaren hiru ikerketetatik lortutako emaitzak aurrezten ditu.

Horrera, maila ezberdinetako programatzaileentzat, aditu ez hain adituentzat, programen arazketako jokaeran eredu bat proposatzen du.

ABSTRACT

This thesis shows the result obtained in tree experiments, carried out in an academic environment, on the debugging process followed by programmers.

Also, a behavioural model is proposed for the debugging process, followed by the different groups of programmers: beginners, those with no experience, intermediate experience and finally experts in the art of programming.

I N D I C E

CAPITULO 2 : COMPORTAMIENTO EN LA DEPURACION DE PROGRAMAS

CAPITULO 3 : UN ANALISIS DE PROTOCOLO DEL PROCESO DE DEPURACION

CAPITULO 4 : DIFERENCIAS EN LA COMPRENSION DEL PROGRAMA ENTRE PROGRAMADORES EXPERTOS E INEXPERTOS

CAPITULO 5 : UN MODELO DE COMPORTAMIENTO EN LA DEPURACION PARA PROGRAMADORES EXPERTOS Y NOVATOS

CAPITULO 1

INTRODUCCION

La depuración de programas, consiste en la localización y corrección de los errores, que un programa de ordenador puede tener, es una de las tareas mas usuales en el mundo de la programación.

Es bastante normal que una parte muy significativa del tiempo que invierte un programador en el desarrollo de un programa, se destine a la depuración de dicho programa; este tiempo se estima del orden de un 40 a 50 % sobre el tiempo total de desarrollo [Boeh81], [Glas82], [Zelk80], [Myer79]. Sin embargo en la actualidad se sabe todavia relativamente poco, sobre la forma de depurar programas.

Depurar programas de ordenador es dificil por varias razones:

a.- Los modelos conceptuales que desarrollan los programadores sobre la forma en que se comportan sus programas, frecuentemente estan equivocados y por eso hay errores que no perciben.

b.- Las herramientas existentes para depurar programas, son a menudo dificiles de usar, o por lo menos miradas con desconfianza por los programadores, con lo que o no se utilizan o se utilizan de forma poco óptima.

c.- La semántica del programa, cambia a medida que los programadores localizan y corrigen los errores. Esto hace que les sea dificil mantener la comprensión del comportamiento del programa.

d.- Los programadores deben seguir simultáneamente varios aspectos del procedimiento del programa [Goul75]. A menudo esto incluye el flujo de datos entre los procedimientos, el flujo de control del programa y las entradas y salidas del mismo.

e.- La depuración requiere una aplicación efectiva de los métodos de resolución de problemas, el conocimiento del lenguaje de programación, el conocimiento del dominio del problema y las técnicas de depuración aprendidas a base de experiencia.

1.1. Razón para el Estudio de la Depuración

Hay dos motivos básicos para el estudio del proceso de depuración de un programa:

El primero, es que no se domina mucho el proceso de la depuración. Se piensa que la depuración de programas es una habilidad adquirida a través de la experiencia. Se han publicado varios libros [Brow 73], [Tass78], [Ward86], dedicados enteramente a la depuración y en la mayoría de cursos de programación y libros de texto, se incluyen algunas técnicas y estrategias de depuración. Sin embargo, ninguno de estos libros se basa en datos empíricos y no hay ninguna taxonomía de las técnicas de depuración aceptadas en general, ni tampoco una serie de reglas que describan cuándo una determinada técnica debería ser aplicada. Además, no hay métodos objetivos para determinar si alguien es un buen depurador o no.

El segundo motivo del estudio de la depuración, es el deseo de desarrollar ayudas útiles para la depuración de programas. Puesto que la depuración ocupa una gran porción del tiempo de un programador, se necesitan medidas preventivas para minimizar el tiempo de depuración durante el ciclo de desarrollo. Una manera de llevar a cabo esto es diseñando y desarrollando instrumentos que permitan a un programador obtener la información necesaria durante la

depuración. Por ejemplo, se ha demostrado que es muy útil para localizar y corregir algunos errores, poder disponer del número de la línea de la sentencia en la que ocurre el error [Goul75], poder disponer del program slicing [Weis82] (descomposición del programa en módulos) y disponer de la naturaleza, del tipo de error y de los valores de las variables que son críticas en el programa. Por todo ello se considera muy útil, experimentar sobre los comportamientos mentales en la depuración, para llegar a saber, que clases de información necesitan realmente los programadores para facilirar su labor y que estrategias de depuración utilizan. La mayoría de los instrumentos para la depuración comercialmente disponibles, se diseñan de una forma ad hoc. Se cree que los depuradores de programas que se basan en el comportamiento mental, proveerán de la información realmente útil a los programadores, y de este modo incrementarán la eficacia de su depuración.

Con estas consideraciones en mente, se describen en esta tésis tres experimentos de depuración, basados en el método de análisis de protocolo.

1.2. Investigaciones Anteriores

La mayoría de los estudios sobre depuración, se han concentrado en la influencia de las características de los programas y en las técnicas de programación utilizadas. Una lista parcial de éstas, incluye las ayudas en la depuración [Goul75], [Shne77], [Broo80], [Oman88], las características del "estilo" del programa [Shne77], las características del lenguaje de programación, [Gann77], [Curt79], y las prácticas de programación [Sack 68], [Myer78]. En estos estudios la tarea habitual de los sujetos era, localizar, o localizar y corregir un único error insertado en el programa y lo que se media era si el error se identificaba correctamente y si era corregido y en cuanto tiempo lo conseguían. Estos estudios casi no proporcionaban información, sobre lo que ocurría en el proceso de depuración, qué estrategias de depuración se utilizaban y cómo, cuándo y dónde se aplicaban. Cinco estudios recientes de protocolo, [Vess85], [Guge86], [Kess86], se han concentrado en el comportamiento de los sujetos durante el proceso de depuración, en un esfuerzo por descubrir, por qué los expertos son más rápidos, más eficientes, encuentran más errores, e introducen bastantes menos errores al hacer modificaciones, que los principiantes.

Aunque estos estudios suministran alguna información general sobre el proceso de depuración y señalan diferencias entre programadores expertos y novatos, no se pueden generalizar los resultados a través de diferentes métodos de depuración, por varias razones. Casi todos los estudios los recientes y los anteriores investigaron el comportamiento de los programadores en la depuración de un programa, que tenía un único error. En muchos de estos estudios los sujetos usaban un método de depuración de papel y lápiz y se les permitía que utilizaran las ayudas de depuración ON-LINE. Sin embargo las condiciones experimentales en estos estudios no se aproximaban a las de las verdaderas situaciones para la depuración.

1.3. Alcance de la Tesis

En esta tesis, investigamos el proceso de depuración de programas, realizados por programadores estudiantes principiantes, de nivel medio y expertos. En general, se presenta un informe de los principales resultados obtenidos en tres experimentos de depuración, realizados durante un periodo de 6 meses en un entorno académico, y se propone un modelo cognoscitivo del comportamiento de programadores expertos y principiantes en depuración. Una característica principal que distingue este estudio de otros estudios previos, es que es muy real el entorno de la depuración usado. Los tres experimentos recogidos aquí se basan en

un método de análisis de protocolo. La técnica de análisis de protocolo, consiste en presentar a un sujeto un problema y decirle que "piense en voz alta" mientras resuelve el problema [Eric80]. Este protocolo verbal se analiza después, en un intento de obtener una traza de los pasos de la resolución del problema, seguidos por el sujeto. A través de un análisis minucioso, a menudo se puede desarrollar una explicación coherente de los pasos empleados por el sujeto, para la resolución del problema.

En muchos métodos de resolución de problemas se reconoce el análisis de protocolo, como un análisis superior a otros métodos, como por ejemplo el de informe retrospectivo. También, se ha empleado este método con éxito, para estudiar el comportamiento de los sujetos en la depuración de programas, [Vess85], [Guge86]. La aplicación con éxito del análisis de protocolo en estudios anteriores, fue la motivación principal para utilizar dicho método en este estudio. En los experimentos relatados en esta tesis, el protocolo verbal de los sujetos fue recogido, sólo mientras se estaba en proceso de edición y durante todas las actividades con el terminal, con anotaciones del tiempo utilizado. El resto de esta sección describe brevemente los tres experimentos y algunos aspectos principales de los resultados.

En el primer experimento los programadores expertos, los de nivel medio y los principiantes, utilizaron el ordenador para depurar un programa escrito en Pascal, que tenia tres errores semánticos y tres errores lógicos. A los programadores se les dió un listado impreso del programa defectuoso, una copia impresa del fichero de datos de entrada, y una copia impresa de la salida correcta. Los resultados indicaron, que los expertos eran más eficientes, ya que encontraron los errores más fácilmente, probaron menos hipótesis y no introdujeron errores, producidos por ellos al hacer modificaciones. Los expertos corregian varios errores antes de verificar las correcciones que iban haciendo, mientras que los novatos y los de nivel medio corregian y verificaban los errores de uno en uno. La mayoria de los novatos corrigieron primero los errores semánticos y después los errores lógicos, mientras que los expertos corregian los errores semánticos y los lógicos al mismo tiempo. La mayor diferencia entre los expertos y los principiantes estaba en la estrategia empleada en la depuración. Parecia que los expertos aplicaban una estrategia de comprensión, por la cual primero trataban de entender lo que el programa hacia y después utilizaban este conocimiento para localizar y corregir errores. Los programadores de nivel medio y los novatos parecia que empleaban un acercamiento aislado, por el que inmediatamente intentaban identificar los lugares del posible error, usando la salida para obtener más pistas, volviendo a incurrir en errores similares.

En el segundo experimento, diferentes grupos de programadores expertos y principiantes, trataron de depurar un programa Pascal que tenía tres errores lógicos. A los programadores, se les dió un listado impreso del programa defectuoso, una copia impresa del fichero de los datos de entrada, y copias de las salidas correctas e incorrectas. Durante la sesión de depuración, después de que los programadores hicieran su primera modificación en el programa, todos los materiales fueron recogidos y se les pidió que reconstruyeran el programa.

El tercer experimento era similar al segundo, excepto en que la tarea de reconstrucción del programa se efectuaba 12 minutos después de haber efectuado la primera depuración del programa.

El propósito de los dos últimos experimentos era, el de probar la hipótesis de que el nivel de comprensión del programa, es un buen vaticinador del rendimiento total de un sujeto, a la hora de la depuración. Los resultados demostraron, que la comprensión del programa juega un papel muy importante en la localización y corrección de los errores lógicos del programa, pero por el contrario es poco significativa, para corregir los errores de tipo semántico. Los programadores expertos tenian más éxito en corregir errores, porque primero intentaban comprender el programa, antes de hacer ninguna modificación en el programa defectuoso, los novatos sin embargo, sin comprender

totalmente el programa intentaban corregir un error aislado, que además les costaba reparar, porque aislaban incorrectamente el error. Por todo ello se insiste en que el nivel de comprensión del programa por parte de los sujetos durante la depuración, parece ser un factor importante en el rendimiento total de la depuración, por lo menos para los errores lógicos.

El modelo cognoscitivo propuesto en esta tesis considera que el comportamiento de los programadores en la depuración, tiene mucho que ver con un proceso de comprensión del programa. Este modelo refleja el comportamiento de los expertos y principiantes en la depuración y proporciona una descripción detallada de los procesos llevados a cabo por los programadores e identifica las principales diferencias existentes a la hora de depurar, entre programadores expertos y principiantes.

Esta tesis consta de cuatro partes. La primera parte, el capítulo 2, proporciona un repaso extensivo de la literatura existente sobre, investigación del comportamiento a la hora de la depuración de programas e identifica cuestiones importantes, relacionadas con la metodología de la depuración experimental y la teoría de la depuración. La segunda parte, capítulo 3, informa, sobre los resultados del primer estudio de protocolo, que comparó el proceso de depuración entre programadores expertos, de nivel medio y aprendices. La tercera parte,

capítulo 4, informa, sobre dos experimentos, en los que se investigan las diferencias de comprensión del programa en la depuración, por parte de programadores expertos y novatos. La cuarta parte, capítulo 5, propone, un modelo descriptivo del comportamiento de programadores expertos y aprendices en la depuración de programas. Las conclusiones y los futuros trabajos se tratan en el capítulo 6.

CAPITULO 2

INVESTIGACION DEL COMPORTAMIENTO EN LA DEPURACION DE PROGRAMAS: UNA VISION GENERAL.

2.1. Introducción

La depuración es el proceso por el que se consigue, que un programa se comporte como está destinado a comportarse. La diferencia entre comportamiento propuesto (correcto) y comportamiento real (incorrecto), viene dada por varios tipos de errores, que el programa puede tener y que han de ser localizados y corregidos durante el periodo de depuración.

Muchos estudios [Boeh81], [Glas82], [Zelk80], [Myer79], dan cuenta de que la depuración de los programas, se lleva casi la mitad de los costes de desarrollo de programas. A pesar de su importancia en el ciclo de desarrollo, se ha investigado poco este tema.

Los principales fines de este capítulo son:

a.- Hacer una revisión extensa de la literatura de investigación del comportamiento, existente sobre la depuración de programas.

b.- Tratar algunas cuestiones importantes relacionadas con la metodología experimental de la depuración, la teoría de la depuración y el problema de la generalización de los resultados obtenidos.

c.- Tratar sobre algunos problemas de la investigación experimental que necesitan ser comentados.

2.2. Objetivos

Hay tres objetivos globales para este resumen:

El primer objetivo es identificar los diferentes métodos de investigación utilizados en los estudios sobre la depuración.

El segundo objetivo es clasificar los estudios sobre depuración, según el método de investigación y describir los principales resultados de estos experimentos.

El tercer objetivo es clasificar el entorno de la depuración utilizado en los estudios experimentales e identificar algunos de los problemas que se le asocian.

2.3. Métodos de Investigación

La selección de un paradigma de investigación apropiado, es crítica en cualquier investigación científica. Un vistazo superficial a la literatura de investigación en la depuración de programas indica, que los métodos empleados van desde estudios de protocolo de un sujeto, hasta experimentos bien diseñados y controlados e incluso hasta extensos estudios de campo. Lukey, en una revisión de la comprensión y depuración del programa, identifica tres métodos para estudiar habilidades de programación sistemáticamente: enfoque experimental, enfoque de inteligencia artificial y enfoque observacional.

> "El enfoque experimental, supone construir una teoría y probarla experimentalmente. El enfoque de inteligencia artificial ,supone construir una teoría y probarla, diseñando un sistema de ordenador, que incorpora esa teoría. El enfoque observacional, supone reunir e

> interpretar los datos sobre el comportamiento que el programador tiene de forma natural"[Luke80].

La revisión aquí presentada se concentra exclusivamente en el enfoque experimental.

En un estudio de los métodos formales de investigación dentro de las cuestiones del diseño del lenguaje, se identificaron tres modos de investigación formal del enfoque experimental: experimentación controlada, análisis de protocolo y estudios de campo. La experimentación controlada, intenta descubrir las relaciones causa-efecto, en ella el experimentador, manipula cuidadosamente las variables independientes y observa los efectos de estos cambios en las variables dependientes, bajo condiciones controladas. En un estudio de protocolo se le presenta un problema a un sujeto y se le pide que "piense en voz alta" mientras resuelve el problema [Eric80]. En los estudios de protocolo, el experimentador mantiene un registro escrito o una cinta grabada de los procesos de pensamiento percibidos de los sujetos o una grabación en video de todos las operaciones efectuadas con el terminal por los mismos. Estos datos verbales o visuales pueden ser revisados y analizados para ver la frecuencia de ciertas actividades o grupos de paradigmas de comportamiento. El estudio de campo es un estudio de investigación experimental que se dirige hacia un marco de la vida real. El experimentador

manipula activamente las variables, controla cuidadosamente la influencia de tantas variables superfluas como la situación lo permita y recoge las observaciones o de los intervalos de tiempo fijo o de los de tiempo casual.

Aunque los experimentos controlados y los estudios de campo controlan mejor las variables experimentales, no son adecuados para estudiar la depuración. Para la actividad de depuración, el análisis de protocolo es superior a esos otros métodos de reunir datos experimentales, porque a través de un análisis cuidadoso, es posible desarrollar una explicación coherente de los métodos o estrategias de depuración empleados por los sujetos. La mayor ventaja del análisis de protocolo, cuando lo comparamos con estos otros métodos, es que con él se reune toda la información que el sujeto puede verbalizar y comunicar a través de sus actividades y gestos mientras ocurre.

El análisis de protocolo tiene sus inconvenientes. Uno es que el requerimiento de "pensar en voz alta" tiende a retardar la marcha de los sujetos. Sin embargo, [Eric80], ha mostrado que esto no afecta al orden o al contenido de sus pasos en la resolución del problema. Otra desventaja es que no se graba qué ocurre durante la elaboración,cuando los sujetos piensan sobre el problema durante un tiempo y llegan a una solución. A pesar de estas limitaciones, el análisis de protocolo es el mejor método conocido para recoger los pensamientos de alguien.

2.4. Estudios de Depuración Experimental

En esta sección, analizamos los estudios experimentales sobre factores de programación que se cree que afectan a la depuración. Los estudios están organizados por el tipo de investigación: experimentos controlados, análisis de protocolo y estudios de campo.

2.4.1. Experimentos Controlados

La experimentación controlada es con mucho, el método más popular para el estudio de la depuración. Por esto, esta sección está subdividida según los factores de programación estudiados.

2.4.1.1. Prácticas de Programación

Probablemente uno de los primeros experimentos controlados sobre el rendimiento total del programador, fue el estudio de [Sack68], de depuración en configuraciones de tiempo compartido y proceso por lotes. Dirigió dos experimentos de sondeo para comparar el rendimiento total de la depuración en programadores, que trabajaban bajo condiciones de acceso directo al ordenador y de forma autónoma. En su primer experimento, 12 programadores

expertos codificaron y depuraron 2 programas, utilizando recursos on-line de depuración o medios no interactivos, mientras que en su segundo experimento, 9 programadores aprendices hicieron la misma tarea, utilizando recursos interactivos o no interactivos también. Los resultados favorecieron la condición interactiva pero sin embargo, este experimento es más famoso por las grandes diferencias de tiempo de depuración (28:1) existentes entre los programadores, a favor de los expertos.

Un experimento en el area de pruebas de programas dirigido por [Myer78], es significativo. Cincuenta y nueve profesionales con gran experiencia en programación, fueron divididos en tres grupos y se les pidió que probaran un pequeño programa PL/1, que contenia 15 errores conocidos. A los sujetos del primer grupo se les pidió probar el programa utilizando un terminal y las especificaciones. Los sujetos del segundo grupo usaron no sólo un terminal y las especificaciones, sino que también usaron una copia del listado del programa. Los sujetos del tercer grupo fueron divididos en equipos de tres personas y se les dió las especificaciones y el listado del programa. Además, a los sujetos del tercer grupo se les pidió, probar el programa utilizando el método de simulación manual . No se les dijo el número de errores, ni la naturaleza de los mismos a ninguno de los sujetos de los tres grupos. Al final de la sesión de prueba, los sujetos describieron los errores que encontraron y anotaron el tiempo utilizado en

prepararse para la prueba y el tiempo empleado en la prueba. Los resultados de este experimento fueron:

a.- Todos los métodos de prueba son iguales en términos de capacidades de detección de errores.

b.- Ciertos tipos de errores son muy difíciles de detectar.

c.- La habilidad de detectar ciertos tipos de errores varía según el método.

d.- Había una variabilidad significativa entre los sujetos, tanto por el número de errores encontrados como por el tipo de los mismos.

2.4.1.2. Ayudas en la Depuración (Debugging Aids)

Los programadores utilizan muchas fuentes de información diferentes, tales como los mensajes de error del compilador y del sistema, listado de los datos de entrada, listado incorrecto de salida, diagramas de flujo y conocimiento del dominio para depurar programas. Varios estudios, [Oman88], [Shne77], [Broo80], observaron la influencia de esta información en la depuración de programas. Gould, pidió a un grupo de profesionales experimentados, que encontraran el error que podía ser de

tres tipos : matriz (array), iteración y sentencia de asignación, que habia en un programa FORTRAN de una página. Se estudiaron varios tipos diferentes de ayudas de depuración: el listado del programa, el indicador del tipo de error, el número de línea donde ocurre el error, los datos de entrada, más la correspondiente salida incorrecta y los datos de entrada más los datos de salida incorrectos y los correctos. Sus resultados mostraron que los errores de asignación se tardaba cuatro veces más en encontrarlos y además se encontraban menos frecuentemente que los otros dos tipos de errores. Notaron que los errores en las sentencias de asignación eran los más difíciles de detectar, principalmente porque los sujetos generalmente tenian que entender el programa para encontrar el error. Una vez que localizaban el error, los sujetos eran capaces de repararlo en casi todos los casos. La depuración era mucho más rápida si el programador tenia experiencia previa con el programa. Demostraron que también el número de línea de la sentencia donde el error ocurría, era con mucho la mejor ayuda en la depuración [Goul75].

Gould, pidió por otra parte a 10 programadores profesionales, con al menos 4 años de experiencia, que identificaran la línea que contenía el error en los mismos 12 programas usados en el anterior estudio. A los sujetos se les dió un listado del programa y la salida y se les dijo que podían utilizar una herramienta de depuración si

lo deseaban. Se dió cuenta de que los sujetos que eran rápidos depurando, tendían a encontrar más errores y cometían menos errores añadidos, que los sujetos que eran lentos depurando. Se encontró también con grandes diferencias individuales entre los programadores, que eran similares a las encontradas en el estudio de Sackman, pero en este estudio las diferencias individuales eran considerablemente menores que la proporción 28:1 hallada por Sackman. Sorprendentemente, los sujetos utilizaron las herramientas de depuración sólo en el 15% de los programas [Goul75].

Oman, comparó las habilidades de programadores estudiantes principiantes, de nivel medio y expertos en la depuración. En particular se concentró en cómo la experiencia en programación y los mensajes de error afectan a la habilidad de depuración. Se les presentaron 2 programas PASCAL a los sujetos y se les pidió que encontraran y corrigieran un único error (error de límites de matriz o un error indefinido) en cada programa. La única pista sobre el error, era el mensaje de error en sí mismo; los sujetos no tenían la salida real para compararla con la salida esperada y no se les permitía usar herramientas de depuración. La mitad de los sujetos recibieron un mensaje de error conteniendo el número de línea y sólo la mitad del mensaje de error. Los programadores experimentados localizaron el error más rápido e hicieron la corrección apropiade más a rapidamente

que los programadores con menos experiencia y además el rendimiento total de los expertos era casi idéntico tanto para los mensajes de error que tenían número de línea como para los que no lo tenían. Los programadores de nivel medio rindieron un poco mejor cuando el mensaje de error contenía el número de línea. Los aprendices dieron resultados pobres comparado con los expertos y los de nivel medio, cuando el mensaje no contenía el número de línea [Oman88].

Shneiderman, dirigió una serie de experimentos para evaluar el uso de diagramas de flujos detallados en tareas comunes de depuración tales como la elaboración del programa, la comprensión, la depuración y modificación. En su experimento de depuración,setenta estudiantes de un curso de nivel medio de programación FORTRAN, depuraron un programa FORTRAN que tenía 147 lineas de código fuente, que tenía tres errores no sintácticos. Los sujetos recibieron un listado del programa defectuoso, la salida incorrecta y o bien, un pequeño diagrama de flujo, o bien un diagrama de flujo detallado o ningún diagrama de flujo. Se les dijo que los programas tenían al menos un error, que era evidente en la salida. No se encontró ninguna diferencia significativa entre los grupos con y los grupos sin, diagramas de flujo. Los experimentadores también sugirieron que "si alguien depura con éxito un programa esa persona debe comprenderlo minuciosamente" [Shne77].

Brooks, también dirigió un experimento para consignar el problema de si la presentación de la información del procedimiento en forma de diagramas de flujo, ayuda o no en la localización y corrección de errores. Se dió a 24 investigadores postgraduados y voluntarios, instrucciones del procedimiento correcto que un ordenador hipotético debería seguir, para controlar una bomba de aceite multigrado. Un grupo de los sujetos recibió una descripción procedimental de las instrucciones en forma de diagrama de flujo, mientras que el otro grupo lo recibió en forma de listado. A los sujetos se les mostró un listado de 56 errores que ocurrirían cuando se siguieran procedimientos incorrectos en el programa. Después de estas instrucciones iniciales se dió a los sujetos 3 procedimientos defectuosos, junto con un síntoma de cada uno de estos procedimientos y se les pidió que identificaran cuál de los listados de 6 errores se correspondía con los síntomas. Tres tipos de errores (cálculo incorrecto, paso perdido en el cálculo e incorrecta secuencia en una sentencia condicional) fueron seleccionados en las areas diferentes de los procedimientos. Los resultados indicaron que el diagrama de flujo falló en mejorar el rendimiento total en la detección del error. Concluyeron diciendo que "la presentación de la información del procedimiento bajo la forma de diagrama de flujo, no ayuda en la correcta identificación de fallos en el procedimiento" [Broo80].

2.4.1.3. Estilo de Programación

Las lineas directrices del "estilo" de programación incluyen cosas tales como los nombres de variables, indentación y comentarios. Shneiderman, llevó a cabo una serie de experimentos diseñados para investigar la eficacia de estas líneas directrices de estilo. En un experimento, los programadores estudiantes de nivel medio, fueron divididos en dos grupos y a cada grupo se le pidió que depurara dos programas PASCAL. Un grupo recibió un programa de búsqueda secuencial, que utilizaba nombres mnemotécnicos (nombres significativos), seguido de un programa de búsqueda binario recursivo que no utilizaba nombres mnemotécnicos, mientras que el otro grupo recibió una versión no mnemotécnica del programa de búsqueda secuencial, seguido de un programa de búsqueda binario recursivo mnemotécnico. A los sujetos se les dijo que cada programa contenia sólo un error. No se encontraron diferencias significativas entre estos dos grupos en el tiempo de búsqueda del error. Sin embargo, Shneiderman concluyó que para programas más complejos, los nombres de variables mnemotécnicos ayudaron en la comprensión del programa por parte de los sujetos. En otro experimento utilizando indentación, programadores estudiantes de nivel medio, fueron divididos al azar en dos grupos y se les pidió, que depuraran dos programas PASCAL. Como en el experimento anterior, un grupo recibió una forma con margen de un programa de búsqueda binario iterativo, seguido de

un programa de fusión sin margen, mientras que el otro grupo recibió un programa de búsqueda binario sin margen, seguido de un programa de fusión con margen. A los sujetos se les dijo que cada programa contenía un único error. Los resultados no mostraron diferencias significativas entre las puntuaciones medias de la depuración. Por todo ello él dijo que "la ventaja de la indentación podría no ser tan grande como algunos creían". Sin embargo, él también notó, que las medidas subjetivas de los programadores favorecen la indentación [Shne77].

2.4.1.4. Características del Lenguaje

Gannon dirigió un experimento para evaluar las características de dos lenguajes de programación, TOPPS y TOPPS II. TOPPS II alterando 9 características tales como el orden de evaluación de la expresión, el operador de asignación y sentencias de control. En particular, comprobó la conjetura: "Los errores imputables a una característica (sentencia de asignación) en TOPPS II serían menos frecuentes y/o menos severos que los errores imputables a una característica comparable (control de asignación) en TOPPS". Los sujetos para este estudio fueron 51 estudiantes universitarios entre licenciados y licenciados. Cada sujeto programó soluciones para dos problemas. La mitad de los sujetos, resolvió el problema en TOPPS y la otra mitad de los sujetos lo resolvió en

TOPPS II. A todos los sujetos se les pidió, que presentaran listados intermedios de sus programas. Se examinó el listado de los sujetos que completaron ambos problemas, para encontrar sus errores y todos los errores fueron clasificados de acuerdo con el número de errores, la sucesión del error y la persistencia del error. Basándose en el análisis del error Gannon, concluyó diciendo que los sujetos preferían el uso de la operación de asignación como sentencia [Gann77].

En otro experimento Gannon, comparó la seguridad de los sujetos utilizando un lenguaje "tipado" estáticamente y un lenguaje "no tipado". Treinta y ocho estudiantes fueron divididos en dos grupos y se les pidió que programaran soluciones al mismo problema dos veces. Un grupo de los estudiantes utilizó primero el lenguaje tipado estáticamente y después utilizó el lenguaje no tipado, mientras que el otro grupo usó estos dos lenguajes en orden inverso. Todos los sujetos presentaron el listado de su programa al investigador. Un análisis de los errores cometidos por los sujetos sugirió, que las características de un lenguaje tipado estáticamente aumentaron la seguridad en la programación, más que las características de un lenguaje "no tipado" [Gann77].

Curtis dirigió un experimento para investigar cómo las características del software, se relacionan con la complejidad psicológica del software. A 54 programadores

expertos en FORTRAN, se les pidió que localizaran y corrigieran los errores que había en tres programas a razón de un error semántico por programa. A los sujetos se les dió un listado incorrecto, ficheros de entrada, una salida correcta y una salida errónea y se les permitió trabajar a su propio ritmo. Las cinco variables independientes manipuladas eran: el tipo de programa, la largura del programa, la complejidad del flujo de control, el tipo de error, y la complejidad métrica del soporte de programación. La única variable dependiente medida fué el tiempo de depuración. Concluyó diciendo que la ciencia métrica del software desarrollado por Halstead [Hals77] y la complejidad de la medida ciclomática desarrollada por [Mcca76], estaban relacionadas con la dificultad experimentada por los programadores, en localizar errores en el código [Barr90].

2.4.1.5. Fragmentación del Programa

Weiser, dirigió un experimento para evaluar la teoría de que "los programadores utilizan la técnica de slicing (proceso de fragmentación de un programa) cuando están depurando".

Veintiun estudiantes licenciados en la universidad y programadores profesionales, depuraron 3 programas ALCOL-W en orden aleatorio. El tamaño de los programas era de 75

a 100 líneas de código. En cada uno de estos programas modularizados, se cambiaba una sola sentencia, para provocar un error en uno de los módulos más importantes, que realizaba la mayor parte del cálculo. A los individuos se les dió un listado incorrecto del programa y una muestra de la salida incorrecta. La mayoría de los individuos ejecutaron el experimento en casa. Después de depurar tres programas, se les encargó a los individuos la tarea de identificar los módulos generados según el slicing, utilizados en el programa original entre una serie de 14 fragmentos del código. Los resultados apoyaron la teoría de slicing; esto es, durante la depuración los programadores rutinariamente descomponen los programas en partes coherentes. También sugirió que es más probable, que un programa que no les es familiar, sea fragmentado por los programadores.

En otro experimento Weiser, extendió su trabajo previo sobre program slicing, hacia program dicing (proceso de combinar múltiples módulos) para acotar la localización de los errores del programa. Veinte estudiantes universitarios licenciados y el personal del departamento de programación de una empresa, fueron divididos en dos grupos, grupos de control y experimental. La tarea de los individuos era encontrar tres errores, en un programa FORTRAN. Durante la sesión de prueba previa a la real, el grupo experimental recibió instrucciones sobre el program slicing y dicing y de su uso en la localización de errores

del programa. Antes de la sesión real de depuración, el investigador y los individuos hablan sobre una muestra de la salida correcta, las normativas convencionales sobre nombres de variables, el algoritmo general utilizado en el programa, y una especificación del programa. A los individuos del grupo de control se les dió solo un listado del programa para hacer la depuración, mientras que a los del grupo experimental, se les dió un listado del programa, más un listado de los módulos(slice) relevantes del conjunto de combinaciones de módulos posibles. Se midió el tiempo total de depuración de cada sujeto. El grupo experimental lo ejecutó signitivamente mas rápido, que el grupo de control [Weis86].

2.4.2. Análisis de Protocolo

El análisis de protocolo intenta entender el proceso de depuración. Una seria deficiencia del análisis de protocolo es que el tiempo y el esfuerzo invertidos por cada sujeto, no es representativo cuando el número de sujetos es pequeño, porque excluye un análisis significativo estadístico de los datos. Ha habido pocos estudios del análisis de protocolo en la depuración.

Brooks, estudió el comportamiento de un único sujeto en la codificación del programa y desarrolló un modelo abstracto del comportamiento del programador. En su

estudio, un estudiante licenciado en informática utilizó papel y lápiz y un terminal de ordenador con impresora, conectado a un sistema de cálculo interactivo para construir programas, para resolver 23 problemas cortos, todos ellos incluian la utilización de un array. Se le pidió al sujeto que escribiera el programa en FORTRAN y después lo depurara y procesara. Al sujeto se le pidió también que pensara en voz alta durante la sesión de construcción del programa. Se recogieron tanto los datos del protocolo verbal como visual. Un análisis de los datos del protocolo indicó que el tiempo de escritura del programa que tardó el sujeto, excedió al tiempo de depuración en una proporción de 12 a 1 y el tiempo de depuración y la largura del programa estaban altamente correlacionados. Se llegó a la hipótesis de que esta diferencia se puede imputar al corto tamaño de los programas [Broo80].

Gould, hizo un estudio sobre los hallazgos de un estudio informal del comportamiento en la depuración de 6 expertos y 4 aprendices. Sus expertos eran estudiantes licenciados en informática, mientras que sus aprendices justo acababan de terminar su primer curso. Cada sujeto depuró 2 Programas PASCAL cortos, cada uno de ellos contenia un número de errores sintácticos, semánticos y lógicos. Los sujetos trabajaron con una salida impresa del programa y una salida impresa de los resultados que provienen de los test de ejecuciones. Los expertos

encontraron la mayor parte de los errores y los encontraron más rápido. Se dió cuenta de que los programadores expertos leían el programa en el orden en que sería ejecutado, el programa principal primero,luego los procedimientos solicitados por el programa principal, después los procedimientos solicitados por estos procedimientos, etc.. Esto, aparentemente les permitió formar una comprensión jerárquica del programa, que va de una comprensión total de lo que el programa hace, hasta una comprensión a nivel de sentencia. En cambio, los aprendices leían el programa secuencialmente desde el principio hasta el final como si de un trozo de prosa se tratara. Dada la estructura del PASCAL, esto resultó, en efecto, una lectura del programa en un orden de ejecución incorrecto. Por ello sólo reconocieron lo más simple, los modelos de nivel bajo, tales como incrementar un contador por ejemplo, y tuvieron dificultad en juzgar qué partes del programa eran relevantes y cuáles no para depurar lo que estaban buscando. Por lo tanto, ellos perdieron mucho tiempo simulando algunas partes irrelevantes del programa y no pasaron el tiempo suficiente en otras partes mas importantes. Para localizar un error con varias causas posibles, los expertos eran mejores cambiando de hipótesis según aparecía la evidencia, mientras los aprendices se centraban sólo en una hipótesis cada vez. Debido al escaso número de sujetos, ella no dió una evaluación estadística de sus hallazgos [Goul75].

En un análisis de protocolo verbal de dieciseis programadores profesionales Vessey [Vess86], encontró que la habilidad de unificar partes similares(chunking) era una mejor medida de la pericia depuradora que años de experiencia en programación. Este concepto hace referencia a la capacidad de una persona de organizar los datos en unidades con significado, que ayudan en la resolución del problema. Vessey clasificó a los 16 programadores en dos grupos, uno de ocho aprendices y el otro de ocho expertos basándose en su habilidad de chunking. A cada programador se le pidió que depurara un programa COBOL, que contenía un único error. Ella se dió cuenta de que los expertos tardaban menos tiempo en encontrar y corregir el error, exponían menos hipótesis a cerca del error, y se equivocaban menos. Los expertos trataban de alcanzar una comprensión de alto nivel del programa y de cómo funciona el programa, con la meta de colocar el error en el contexto. Parecía que los expertos permitían que la estructura del programa se abriese, colocando las pistas en el contexto de esa estructura, que el error se conceptualice en términos de la estructura del programa. Los aprendices, por otra parte, parecían más ansiosos por encontrar el error y más entregados a su hipótesis del error, que los llevaría a una búsqueda en profundidad del mismo. Sin un molelo de la función y estructura del programa, los aprendices cambiaban frecuentemente, las posiciones de referencia del programa y les llevaba más tiempo confirmar o desechar las correcciones del error.

Por tanto Vessey concluía, diciendo que mientras los dos grupos usaron esencialmente los mismos métodos básicos de depuración, los expertos eran más efectivos en su aplicación.

Kessler, hizo un informe de una investigación del comportamiento de programadores aprendices, depurando funciones LISP. Pidió a ocho programadores aprendices de LISP, que depuraran 2 secuencias de funciones LISP de una linea. Cada una de estas secuencias contenia 6 funciones erroneas y 3 funciones correctas (servian para despistar). Cada secuencia consistia en 6 errores (2 sintácticos y 4 semánticos) en un orden fijado. A los sujetos se les pidió que contaran en alto, la resolución del problema durante el proceso de depuración. Además del protocolo verbal, las interacciones con el terminal de los sujetos se recogieron anotando el tiempo. El análisis de los datos del protocolo indicó que los sujetos tardaron tanto en completar el primer problema de la secuencia, como para el resto de ellos. Sin embargo, el rendimiento total del sujeto en la depuración, mejoró considerablemente cuando los errores se repitieron en la segunda secuencia. La mayoria de los sujetos utilizaron la misma estrategia general de depuración de 4 fases para todos los problemas: comprensión del código, detección del error, localización del error y corrección del error. Las dos últimas fases contenían la mayor parte de sus datos de protocolo [Kess86]. Finalmente, Kessler y Anderson desarrollaron un

modelo de sistema de depuración basado en reglas, para aprendices.

Gugerty, dirigió dos experimentos para investigar las diferencias entre expertos y aprendices en la depuración. El pidió a programadores estudiantes, tanto expertos como aprendices, que encontraran y corrigieran un único error en un programa utilizando depuración en sistemas ON-LINE.

Los sujetos en el primer experimento, depuraron un programa en LOGO, los sujetos del segundo en PASCAL. A través de los datos de protocolo se dieron cuenta de que aunque los expertos y los aprendices emplearon la misma estrategia de depuración, los expertos eran más rápidos y tenían más éxito. Antes de generar y probar las hipótesis iniciales, los expertos y los aprendices distribuyeron sus actividades de la misma manera. Los aprendices tardaron más en depurar los programas porque su hipótesis inicial no era a menudo la correcta y de esa manera tuvieron que generar más hipótesis. Finalmente, concluyeron diciendo que las diferencias de ejecución de expertos y aprendices se debían a la diferente habilidad de comprensión del programa de cada uno de estos grupos [Guge86].

2.4.3. Estudios de Campo

Youngs, estudió los errores en función de la frecuencia del error, la propensión al error, y el diagnostico del error y proporcionó descripciones cuantitativas de los errores para entornos de programación representativos. En su experimento, un grupo de 30 estudiantes universitarios que cursaban el primer curso de programación y 12 programadores profesionales presentaron cuestionarios, cuadernos de carga y todas las salidas del ordenador que llegan de los procesos de depuración. Se utilizaron los lenguajes de programación ALGOL, BASIC, COBOL, FORTRAN, Y PL/1 para programar. Con el fin de abstraer la información suministrada por los sujetos,un esquema de codificación clasificó los errores por tipos de sentencias funcionales (asignación, I/O, iteración, etc...) el tipo de comprensión requerido para corregir el error, la manifestación especifica del error (omisión, operación ilegal, etc...) y respuestas del sistema. Los errores del programa se analizaron trabajando hacia atrás, desde la versión correcta hasta la versión inicial incorrecta. Los resultados principales de este estudio fueron:

a.- Ocho tipos de construcciones de programa contabilizaban el 75% de todos los errores, las asignaciones el 29% y las entradas/salidas el 14%.

b.- El 75% de los errores hechos por los programadores estudiantes, eran errores semánticos y lógicos.

c.- Las sentencias de formato de E/S de ALGOL y FORTRAN, eran la fuente de un gran número de errores.

d.- Los programadores profesionales eliminaron los diferentes tipos de errores a velocidades diferentes, mientras que los aprendices los corrigieron aproximadamente todos a la misma velocidad [Youn74].

2.5. Areas Problemáticas en la Experimentación de la Depuración

Varios estudios [Broo80], [Mohe82], contienen excelentes argumentos, sobre los problemas metodológicos en la aplicación de la experimentación psicológica al estudio de la programación y de los programadores. Recalcan la importancia de los estandares de rigor

metodológico y clasifican las principales cuestiones metodológicas. En esta sección usaremos una clasificación similar en una breve revisión de los problemas metodológicos experimentales en los estudios de depuración.

2.5.1. Los Sujetos

Los sujetos de los experimentos son un componente vital en cualquier experimento de depuración. Sin embargo, son notorias las diferencias entre los programadores. Por eso es dificil dividir los sujetos en grupos experimentales. Empleando un gran número de sujetos y un diseño experimental apropiado, se mitiga este problema en cierta manera.

En los diferentes estudios revisados en este capitulo se clasificaron los sujetos en diferentes niveles de experiencia, según los niveles de educación [Guge86], [Oman88], según "la habilidad de slicing(fragmentación)" de los programadores [Vess85], según la habilidad de programación y según la experiencia en programación [Goul75].

La mayoria de las clasificaciones se basaron según lo que convenia. La clasificación de los estudiantes, la fuente más común de sujetos de experimentación porque constituyen una fuente de trabajo variable y facilmente

disponible, se basa generalmente en el curso que estén realizando y los programadores profesionales se clasifican generalmente según los años de experiencia que tienen. Otro factor que complica la cuestión, es que los programadores varían mucho en su habilidad de ejecutar varias tareas basadas en su experiencia, habilidad natural, motivación y entorno [Curt79].

2.5.2. Materiales

En los estudios de depuración se han usado los lenguajes de programación de alto nivel FORTRAN, PASCAL, LISP, COBOL, ALGOL-W, y LOGO. Puesto que los lenguajes de programación difieren unos de otros en la elección de nombres significativos de variables, en la estructura de los datos, en la documentación de los datos, en la modulación, las estructuras de control, las especificaciones de entrada y salida y las formas sintácticas, los resultados de un estudio se podrían aplicar sólo al lenguaje particular usado en ese estudio. Con el fin de evaluar la generalización de los resultados a través del lenguaje de programación, los investigadores necesitan confrontar los experimentos de depuración, con diferentes lenguajes de programación. Tampoco está claro si los sujetos exhiben el mismo o diferente comportamiento en la depuración, cuando tratan programas realizados en lenguaje procedimental, no procedimental o ensamblador.

La extensión del programa es otro problema. Los programas utilizados en estudios de depuración tienen una largura, que va desde una única línea de función LISP [Vess86], hasta 300 líneas de código FORTRAN [Shep79]. En un estudio sobre depuración Sheppard, investigó los efectos de la longitud del programa, el tipo de error y la estructura de control en la tarea de depuración y llegó a decir lo siguiente:

> "Incrementar el número de líneas de código, tuvo un modesto efecto en el tiempo necesario para localizar y corregir el error. El tiempo requerido para corregir un error del tipo estudiado aquí (semántico- cálculo, lógico y datos) no se incrementa linealmente con el número de líneas del código, entonces parece que hay 2 estrategias distintas para corregir un error - la identificación de la subrutina que contiene el error y la búsqueda del error dentro de la subrutina" [Shep79].

Shneiderman dice que "los programadores que trabajan en programas largos, tienen la posibilidad de aplicar estrategias diferentes a las de los que trabajan en programas cortos" [Shne80], de tal forma que la extensión del programa limita rigurosamente la generalización de las conclusiones sacadas de estos estudios.

El problema de generalización surge de nuevo en la selección del tipo de error utilizado en los experimentos de depuración. Un estudio de la literatura muestra que los errores van desde errores sintácticos,hasta errores semánticos, de tipo lógico y de tiempo de ejecución. La mayoria de los estudios previos sobre la depuración empleaban errores no sintácticos, porque los errores sintácticos son fáciles de localizar y corregir. Por ejemplo [Goul75], estudió la depuración como una función de la sintaxis de la sentencia que contiene el error, y como una función de la información disponible por el programador. El, terminó diciendo que dar información adicional al programador tenia poco efecto, pero proporcionar el tipo de error si marcaba una gran diferencia.

Gould, estudió la depuración como una función de la posición de la sentencia que contiene el error, en una jerarquia proposicional que representa el programa. El concluyó diciendo que la dificultad de encontrar un error, estaba correlacionada con la posición en la estructura. Gould también atribuyó la naturaleza del error, como el primer determinante de la dificultad del error. Por otra parte posteriormente el mismo, mantuvo una visión diferente de la dificultad de encontrar un error y estableció que:

"Puede que no haya en principio una manera a priori, de categorizar

errores por la dificultad. El determinante primario de la dificultad del error, será el proceso usado para encontrar el error. Este proceso estará fuertemente influenciado, por la cantidad y la naturaleza del contexto usado, para entender el error. Algunos errores se pueden encontrar viendo sólo la zona en la que ocurren (algún error de ortografía), otros requieren un conocimiento de las sentencias que lo rodean (muchos errores sintáticos), otros requieren una comprensión de la sentencia y del papel que juega en el programa. Otros aún, sólo se pueden encontrar entendiendo el algoritmo que soluciona un subproblema y otros incluso requieren una comprensión de varios de los subproblemas y de cómo se interrelacionan. Sin embargo, siempre habrá varias maneras para descubrir cualquier error dado, así que la naturaleza del error en sí mismo, no tiene la posibilidad de ser un buen predictor de los procesos que se usarán para aislarlo; el conocimiento del programador y la

estrategia de depuración si la tienen" [Goul89].

2.5.3. Procedimiento

En todos los experimentos de depuración comentados se les proporcionó a los sujetos un listado del programa defectuoso. Además a los sujetos se les dió material suplementario tal como una especificación detallada del programa, diagramas de flujo, muestras de la salida correcta, un listado de la salida incorrecta, y pistas sobre el tipo y localización del error. Los sujetos tenían opción a usar selectivamente o no usar esta información para localizar y corregir los errores. Sin embargo Gould, dijo:

> "Los sujetos hacen, uso de muchas fuentes diferentes de información para depurar el programa. Además de las fuentes que ellos traen al experimento, tales como la experiencia, aptitud, motivación, conocimiento de la aplicación, y preferencias por diferentes estrategias de depuración, hay fuentes diferentes (datos de entrada, datos de salida, nombres mnemotécnicos,

> comentarios) disponibles en el experimento mismo. Los caminos normales por los que los sujetos llegan a la depuración están relacionados con, qué fuentes de información están disponibles y cuáles se utilizan" [Goul89].

Las instrucciones que se dan a los sujetos durante el experimento de depuración, pueden jugar un papel vital en determinar el resultado del experimento. Por ejemplo, el conocimiento del número de errores en un programa o la proximidad al error dentro de un programa, pueden influir en la selección de una estrategia concreta o en minimizar el tiempo de depuración. Por tanto las instrucciones para la depuración dadas a los sujetos durante la experimentación deben ser precisas y no deben ser ambiguas. Puesto que las instrucciones de la depuración dadas a los sujetos varían de un experimento a otro, es muy difícil comparar y contrastar resultados de estos estudios. Schneiderman, ha señalado esto:

"Los experimentos de depuración, sufren la artificialidad de la experimentación de laboratorio. Si a los sujetos se les dice, que existe un único error que puede ser corregido con la modificación de una sola línea, la situación es enormemente diferente de la realidad, donde tal información no está disponible. Seria interesante estudiar el rendimiento total y reacciones emocionales cuando se incluye un único error en un programa y a algunos se les dice que existe solamente un error, a otros se les dice que existen 5 errores y a otros se les dice que existe un número desconocido de errores (posiblemente cero)" [Shne80].

2.5.4. Variables Dependientes

Las variables dependientes, son un componente importante en cualquier experimentación controlada. En casi todos los estudios sobre depuración, sólo las medidas de tiempo y corrección se usan como variables dependientes. Las medidas típicas de corrección, eran si el error era localizado correctamente o localizado y corregido, el número de errores corregidos, y el número de errores producidos durante la depuración. La medida del tiempo era a menudo el tiempo total de depuración de los programadores. Un problema principal con las medidas del tiempo, era que el investigador no puede garantizar, que todo el tiempo del sujeto fuera empleado en la tarea de depuración. Estas medidas proporcionan las diferencias del rendimiento total en la depuración, entre programadores expertos y aprendices, pero no proporcionan casi nada de información del proceso real de depuración.

2.6. Resúmen

La depuración del programa, es una tarea compleja con muchos componentes que contemplar y relativamente poca evidencia empírica para explicar, lo que los programadores hacen realmente cuando depuran su programa o el programa de otro. En nuestra revisión de los estudios del comportamiento en la depuración, evaluamos varios métodos de búsqueda, discutimos la teoría en si de la depuración e identificamos los inconvenientes de la experimentación de la depuración. Finalmente, se enumeran a continuación, algunas cuestiones relacionadas con la depuración de programas, que no han sido tratadas adecuadamente:

a.- Estrategias de depuración empleadas.

b.- Dificultad de los diferentes tipos de errores de programación.

c.- Clasificación de los sujetos basada en la experiencia en depuración.

d.- Influencia de los factores del programa.

e.- Influencia de los factores del programador,

f.- Ayudas de depuración usadas o necesarias.

CAPITULO 3

UN ANALISIS DE PROTOCOLO DEL PROCESO DE DEPURACION

3.1. INTRODUCCION

Casi todos los estudios de depuración, se han concentrado en las diferencias del rendimiento total, entre programadores expertos y aprendices. La tarea normal del sujeto en estos estudios, era localizar o localizar y corregir un único error insertado en el programa y las medidas del rendimiento total, se efectuaban en base a si el error era correctamente identificado y corregido y el tiempo necesario para conseguirlo. Puesto que sólo se consignaba la corrección y el tiempo invertido, estos estudios no proporcionaron casi información, sobre lo que ocurrió durante el proceso de depuración, qué estrategias se usaron y cómo, cuándo y dónde fueron aplicadas. Sin embargo, 3 estudios sobre el análisis de protocolo [Guge86], [Vess85], investigaron las diferencias del proceso de depuración efectuado por programadores expertos y aprendices. Los resultados principales de estos estudios se explicaron en el Capítulo 2.

En este Capítulo presentamos los resultados de nuestro primer estudio de protocolo, que sirvió para comparar el proceso de depuración, de los programadores estudiantes aprendices, los de nivel medio y los expertos. A los sujetos se les dieron el listado de un programa defectuoso que tenia 6 errores, el archivo de los datos de entrada la salida deseada y el programa grabado,en un diskete y se les pidió que corrigieran el programa defectuoso. Además, a los sujetos se les pidió que pensaran en voz alta, durante cada sesión de edición del programa. Todos los sujetos tenían experiencia previa en entornos de programación Pascal. Intentamos simular el entorno real en la medida de lo posible.

El programa usado en los estudios de [Vess85] y [Guge86], contenia sólo un único error. Nuestro programa sin embargo contenia 6 errores. En el estudio de [Finn87], a los sujetos, se les dió un listado del programa, muestras de ejecuciones de "casos probados seleccionados por el investigador" y si se solicitaba específicamente, las ejecuciones obtenidas por un sistema de depuración interactivo. Nuestros sujetos podían usar libremente el depurador ON-LINE del ordenador o cualquier otra ayuda de depuración y procesar el programa tan a menudo como quisieran.

3.2. EXPERIMENTO 1

3.2.1. Sujetos

Los sujetos de este experimento eran 6 novatos, 6 de nivel medio y 6 programadores expertos en PASCAL, todos voluntarios. Los novatos acababan de terminar un curso de introducción a la programación en PASCAL, de una semana de duración en la Universidad, diseñado especificamente para este experimento. Los de nivel medio estaban terminando su segundo cuatrimestre del curso de programación normal. El grupo de expertos estaba compuesto por estudiantes de informática de último curso, y con experiencia previa en varios lenguajes. Todos los expertos eran programadores estudiantes muy experimentados y, además, incluso alguno de ellos, habia impartido cursos de programación introductorios al PASCAL o habían servido como monitores de otros estudiantes, de cursos de programación inferiores.

3.2.2. Materiales

El programa a depurar era un programa PASCAL que tenía 73 líneas de código en total, que leía 19 registros de un archivo, conteniendo valores enteros,los clasificaba en orden ascendente utilizando el algoritmo de clasificación de la burbuja, y buscaba 5 valores concretos en la lista clasificada, usando una rutina de búsqueda binaria. Todos los sujetos estaban familiarizados con la rutina de búsqueda binaria y el algoritmo de clasificación de la burbuja. El programa se llevaba a cabo incluyendo 2 procedimientos (LEERDATOS y CLASIFICACION) y una función (BUSQUEDA-binaria). El programa principal llamaba a estos tres subprogramas. El programa estaba escrito de una manera estructurada con indentaciones y nombres significativos y contenia 3 líneas de comentarios generales describiendo el programa, pero no tenia comentarios dentro de las líneas ejecutables del programa.

Habia 6 errores - 3 semánticos y 3 lógicos - en el programa. Los programas con errores semánticos (por ejemplo, límites de matriz indebidos, división por cero, etc...) compilan correctamente, pero cuando se ejecutan, terminan anormalmente con un mensaje de error sin mas información. Los programas con errores lógicos tampoco tienen errores en compilación, pero cuando se ejecutan, terminan normalmente aunque con resultados incorrectos. Cada procedimiento y función contenia un error semántico

y uno lógico. Los 3 tipos de errores semánticos eran:

a.- Intentos de leer después del indicador de fin de archivo, en el procedimiento LEERDATOS.

b.- El valor de un índice excede los límites de la matriz en el procedimiento CLASIFICACION.

c.- Incompatibilidad entre los tipos de datos usados en la función de BUSQUEDA-binaria.

Seleccionamos estos errores porque son errores que los estudiantes cometen comunmente. Para los tres errores semánticos, el PASCAL sacó los tres mensajes de error siguientes, junto con un número indicando el número de linea de la sentencia donde el error ocurría:

a.- Se ha hecho un intento de acceder a los datos más allá del fin de un archivo.

b.- El valor de una variable o sub-expresión está fuera de rango.

c.- Se ha encontrado una incompatibilidad entre tipos de datos.

Las tres clases de errores lógicos seleccionados eran:

a.- El número de iteraciones a través de un bucle es incorrecto, en el procedimiento de LEERDATOS,está desplazado en una unidad.

b.- Error de sentencia de asignación - a una variable se le asigna un valor incorrecto en el procedimiento de CLASIFICACION.

c.- Error en predicado - hay una expresión condicional incorrecta en la función de BUSQUEDA-binaria.

El listado del programa defectuoso, que tenía 6 errores, aparece en el Apéndice A1. Cada uno de los errores podría ser corregido modificando una única sentencia. Las versiones correctas de estas sentencias se muestran en el listado del programa defectuoso, como comentarios dentro de líneas ejecutables.

3.2.3. Procedimiento

Los sujetos realizaron la tarea de depuración individualmente. Se les dió un listado impreso del programa defectuoso y una copia impresa del archivo de datos de entrada, ambos estaban también disponibles en un diskette. Se les dió también una copia de la salida correcta (ver Apéndice A2). A los sujetos no se les dijo

cuántos o qué tipo de errores contenía el programa y que cada uno de estos errores podría ser corregido cambiando solo una sentencia para cada uno. Se les informó a los sujetos, de que ellos podrían depurar el programa a su propio ritmo y usar cualquier técnica de depuración o ayudas.

Durante la sesión de depuración, se grabó, qué partes del programa estaban examinando los individuos y que actividades estaban realizando. No se les hizo ninguna pregunta, para confirmar si estaban ejecutando una actividad determinada o no.

Para el objetivo que nos planteamos en este estudio, consideramos las siguientes cuestiones del programa: Los segmentos del programa (es decir, comentarios globales , declaraciones, el programa principal, el procedimiento LEERDATOS, el procedimiento CLASIFICACION y la función de BUSQUEDA-binaria), el listado del fichero de los datos de salida, copia impresa de la salida esperada, listado del programa, ventana del programa PASCAL, ventana del mensaje de error y ventana de la salida del programa. Las actividades ejecutadas por los sujetos fueron categorizadas como sigue:

a.- Examinar el listado de los datos de entrada.

b.- Examinar el listado de la salida esperada.

c.- Examinar el segmento de programa actual(en el listado del programa o en la pantalla).

d.- Examinar la ventana donde aparece el error.

e.- Examinar la ventana de salida del programa.

f.- Segmento del programa simulado manualmente(en el listado del programa).

g.- Meter los datos de entrada.

h.- Usar herramientas ON-LINE de depuración.

i.- Ejecutar el programa.

j.- Modificar el segmento del programa actual.

k.- Comparar la salida esperada con la real.

Después de cada modificación del programa y antes de que ellos ejecutaran el mismo, se les pidió que establecieran su hipótesis sobre los errores. Además, siempre que los sujetos encontraran un error semántico, se

les pidió que explicaran el significado de ese error. El proceso de depuración del sujeto fue grabado, como un resúmen representando el perfil de sus actividades. Podemos poner como ejemplo, el trabajo parcial del sujeto N2, reflejado en el Apéndice A3.

3.2.4. Resultados

La mayoría de los sujetos comenzaron su sesión de depuración estudiando el programa. Las tablas 3.1. muestran, que incluso aunque los expertos tardaron más tiempo en la lectura inicial del programa, su tiempo total de depuración era considerablemente menor al de los novatos. Todos los expertos excepto E1 y E6 pasaron al menos el 25% de su tiempo total de depuración, estudiando el programa antes de comprobar su primera hipótesis, mientras que el 17% fue el tiempo total invertido en el estudio del programa, del que mas tiempo invirtió en esa tarea de los intermedios y el 5% para el de los novatos. De hecho la mitad de los novatos y de los de nivel medio, ejecutaron inmediatamente el programa, sin hacer ninguna lectura inicial del programa.

Datos del Rendimiento Total de la Depuración

Todos los valores que son tiempos, dados en minutos	Número de sujetos						
	N1	N2	N3	N4	N5	N6	Total
Lectura inicial programa	1	0	0	3	2	0	6
Tiempo para 1ª modificación	4	3	3	5	3	6	24
Nº instrucciones cambiadas	14	30	32	18	18	27	139
Nº total de ejecuciones	11	26	24	17	7	21	106
Nº nuevos errores generados	2	10	7	3	1	6	29
Nº de errores no corregidos	3	0	4	3	0	3	13
Tiempo total de depuración	67	66	55	55	35	58	336
Nº write statement puestos	2	5	2	3	1	4	17

Tabla 3.1a

(Aprendices)

Datos del Rendimiento Total de la Depuración

Todos los valores que son tiempos, dados en minutos	Nº de sujetos						
	I1	I2	I3	I4	I5	I6	Total
Lectura inicial programa	3	0	2	4	0	0	9
Tiempo para 1ª modificación	4	5	4	4	3	3	23
Nº instrucciones cambiadas	7	12	8	10	12	13	62
Nº total de ejecuciones	9	16	12	10	15	17	79
Nº errores nuevos añadidos	0	5	1	2	3	3	14
Nº errores no corregidos	0	1	0	0	0	1	2
Tiempo total de depuración	17	52	38	32	31	49	219
Nº write statement puestos	1	3	4	3	3	0	14

Tabla 3.1b

(Intermedios)

Datos del Rendimiento Total de la Depuración

Todos los valores que son tiempos, dados en minutos	Número de sujetos						
	E1	E2	E3	E4	E5	E6	Total
Lectura inicial programa	2	8	7	5	7	0	29
Tiempo de 1ª modificación	7	11	8	8	13	3	50
Nº instrucciones cambiadas	7	6	10	8	7	15	53
Nº total de ejecuciones	4	2	7	5	4	14	36
Nº errores nuevos añadidos	0	0	2	1	0	3	6
Nº errores no corregidos	0	0	0	0	0	0	0
Tiempo total de depuración	14	12	23	20	17	33	119
Nº write statement puestos	0	0	1	1	1	2	5

Tabla 3.1c

(Expertos)

Por otro lado los expertos, los de nivel medio y los novatos diferían también, en el orden en el que leían el programa. Los expertos leían el programa en el orden en el que éste seria ejecutado - el programa principal primero, el procedimiento LEERDATOS a continuación, el procedimiento CLASIFICACION después y finalmente la función de BUSQUEDA-binaria. La mayoría de los novatos y los de nivel medio, leían el programa desde el principio hasta el final secuencialmente, en el orden que se lo encontraban escrito. Esto lo apoyan descubrimientos similares como el de [Finn87], que dice que los expertos alcanzan un alto nivel de comprensión del programa y de cómo funciona, mientras que los novatos usan un acercamiento secuencial para comprender el programa, lo cual hace que tengan un mal entendimiento del mismo. Por ello además los expertos se dirigieron muchas menos veces al programa fuente, que los novatos y los de nivel medio.

Las tablas 3.2. muestran el número de los segmentos de programa, a los que los sujetos se refirieron durante la depuración. En total, los novatos y los de nivel medio se refirieron a segmentos de programa, más a menudo que los expertos (40,2 para novatos, 32,2 para los de nivel medio y 22,8 para expertos). Estadisticamente esto datos son muy significativos respecto a la importancia de la mayor o menor comprensión del programa.

Mas de la mitad de las referencias que hacían los novatos y los expertos afectaban al procedimiento LEERDATOS y a la función de BUSQUEDA-binaria, mientras que más de la mitad de las referencias de los de nivel medio afectaban al procedimiento CLASIFICACIÆN y a la función de BUSQUEDA-binaria.

Todos los expertos localizaron y corrigieron los 6 errores; 2 de los de nivel medio no encontraron el error lógico, dentro de la función de la BUSQUEDA-binaria y de los novatos únicamente 2 hallaron todos los errores. Los novatos introdujeron al menos un nuevo error al tratar de corregir los existentes, 5 de los de nivel medio y 3 de los expertos también introdujeron algún error adicional al hacer modificaciones. Las tablas 3.1, muestran también que los novatos introdujeron 29 nuevos errores durante la depuración y los de nivel medio introdujeron 14 errores nuevos, todos estos errores debidos a las modificaciones que habín hecho al programa, la mayoría de los cuales no fueron corregidos inmediatamente. Los 6 errores introducidos por los expertos, si fueron corregidos inmediatamente.

El número de sentencias depuradas en las tablas 3.1, era el número de sentencias WRITE, que aparecen insertas en distintos segmentos del programa, para seguir la pista de ciertas variables en posiciones clave dentro del programa. Los novatos y los intermedios insertaron muchas

más sentencias WRITE en la depuración, que los expertos. Generaron mucho listado de salida, porque insertaron sentencias WRITE dentro de bucles en posiciones no correctas,con el objeto de imprimir una matriz completa o hicieron la traza de valores de variables no implicadas en el error. Los novatos y los de nivel medio, ejecutaban 4 veces más el programa, que los expertos. Los expertos insertaron bastante menos sentencias WRITE y generalmente lo hacian al final de los procedimientos fuera de los bucles, con lo que no incrementaban excesivamente la salida impresa y además a menudo utilizaron herramientas de depuración ON-LINE, para hacer la traza de las variables. Por el contrario, ninguno de los novatos ni los de nivel medio usaron herramientas ON-LINE. Ellos lo que hacían a menudo era simular manualmente, la ejecución de uno o más de los procedimientos del programa, tomando notas sobre los valores de las variables en distintas fases del programa. Ningún experto ejecutó de forma explícita manualmente el programa.

Frecuencias de las referencias a segmentos de programa

Nº del sujeto	N1	N2	N3	N4	N5	N6	Media	
Declaraciones globales	1	1	2	5	2	3	14	2.3
Programa principal	3	11	7	8	5	11	45	7.5
Lectura de datos	10	8	16	14	5	13	66	11.0
Clasificación por burbuja	5	8	8	16	4	10	51	8.5
Búsqueda binaria	10	14	20	7	6	8	65	10.8
Nº total de referencias	29	42	53	50	22	45	241	40.2

Tabla 3.2a

(Aprendices)

Frecuencias de las referencias a segmentos de programa

Nº del sujeto	I1	I2	I3	I4	I5	I6	Media	
Declaraciones globales	I	6	3	5	2	4	21	3.5
Programa principal	4	3	4	5	6	8	29	4.8
Lectura de datos	3	7	7	5	5	4	31	5.2
Clasificación por burbuja	6	11	3	6	9	18	53	8.8
Búsqueda binaria	5	9	10	16	14	10	64	10.7
Nº total de referencias	19	36	27	37	36	44	199	33.2

Tabla 3.2b

(Intermedios)

Frecuencias de las referencias a segmentos de programa

Nº del sujeto	E1	E2	E3	E4	E5	E6	Media	
Declaraciones globales	3	2	4	3	4	2	18	3.0
Programa principal	7	2	4	5	3	8	29	4.8
Lectura de datos	6	4	6	5	6	6	33	5.5
Clasificación por burbuja	2	3	3	2	3	8	21	3.5
Búsqueda binaria	3	3	7	8	5	10	36	6.0
Nº total de referencias	21	14	24	23	21	34	137	22.8

Tabla 3.2c

(Expertos)

Los expertos, intermedios y novatos se diferenciaban también en el orden y en la manera en la que corregían los errores. Los novatos (excepto N5) y los de nivel medio corrigieron sólo un error cada vez, mientras que los expertos corregían grupos de errores simultaneamente.

Como se muestra en la tabla 3.3.c. el sujeto E1 simultáneamente corrigió primero el error semántico I y los errores lógicos 2 y 3, luego corrigió el error semántico 3 y el error lógico 2; y finalmente corrigió el error semántico 3. Los expertos hicieron las correcciones rapidamente,una detrás de otra y luego ejecutaron el programa para verificar las correcciones que habin efectuado. La tabla 3.3.a. muestra, que los 4 novatos que no habían encontrado todos los errores, fallaron al menos en localizar un mínimo de 3 errores. Estos 4 novatos (excepto N3) localizaron sólo errores semánticos. La tabla 3.3.b. muestra que con pocas excepciones, los intermedios corrigieron primero los 3 errores semánticos y luego los 3 errores lógicos. Nos dimos cuenta de que los sujetos I2 e I6, no corrigieron el error lógico en la función de BUSQUEDA-binaria. Los grupos de errores corregidos a la vez por los expertos,incluyeron errores semánticos y lógicos y a menudo implicaron a los 2 procedimientos y a la función. El tamaño de estos grupos iba desde uno a cinco errores. Los expertos generalmente parecían corregir errores procedimiento por procedimiento (ver sujetos E4 y E5).

Orden de corrección del error

Nº del sujeto	N1	N2	N3	N4	N5	N6
Semántico1	1	1	1	1	3	1
Semántico2	2	2	2	2	1	2
Semántico3	3	3	*	*	4	3
Lógico1	*	6	*	3	5	*
Lógico2	*	5	*	*	1	*
Lógico3	*	4	*	*	2	*

Tabla 3.3a

(Aprendices)

* no pudo corregir ese error

Semántico1:Error en procedure LEERDATOS

Semántico2:Error en procedure CLASIFICACION

Semántico3:Error en función BUSQUEDA-binaria

Lógico1 :Error en procedure LEERDATOS

Lógico2 :Error en procedure CLASIFICACION

Lógico3 :Error en función BUSQUEDA-binaria

Orden de corrección del error

Nº del sujeto	I1	I2	I3	I4	I5	I6
Semántico1	1	1	1	2	1	1
Semántico2	3	2	2	3	2	2
Semántico3	4	3	4	5	3	4
Lógico1	2	5	6	6	6	5
Lógico2	5	4	3	1	5	3
Lógico3	6	*	5	4	4	*

Tabla 3.3b

(Intermedios)

* no pudo corregir ese error

Semántico1:Error en procedure LEERDATOS

Semántico2:Error en procedure CLASIFICACION

Semántico3:Error en función BUSQUEDA-binaria

Lógico1 :Error en procedure LEERDATOS

Lógico2 :Error en procedure CLASIFICACION

Lógico3 :Error en función BUSQUEDA-binaria

Orden de corrección del error

Nº del sujeto	E1	E2	E3	E4	E5	E6
Semántico1	3	2	2	1	2	1
Semántico2	1	1	1	2	1	2
Semántico3	2	1	3	3	1	3
Lógico1	1	1	4	1	2	4
Lógico2	1	1	1	2	1	4
Lógico3	2	1	4	3	1	5

Tabla 3.3c

(Expertos)

* no pudo corregir ese error

Semántico1:Error en procedure LEERDATOS

Semántico2:Error en procedure CLASIFICACION

Semántico3:Error en función BUSQUEDA-binaria

Lógico1 :Error en procedure LEERDATOS

Lógico2 :Error en procedure CLASIFICACION

Lógico3 :Error en función BUSQUEDA-binaria

La estrategia general de corrección de un único error para los novatos y los de nivel medio, consistía en realizar repetidamente los 3 pasos siguientes:

a.- Formar una hipótesis sobre el error.

b.- Modificar una o más sentencias del programa para comprobar la hipótesis.

c.- Ejecutar el programa para ver el efecto de los cambios.

Cuando las sentencias modificadas introdujeron errores adicionales, 4 novatos y 3 de nivel medio, no restauraron inmediatamente el programa a su estado original. La estrategia de la corrección de errores múltiples de los expertos, seguía también los 3 mismos pasos. Sin embargo, su hipótesis versaba sobre errores múltiples y su edición del programa era más extensa. Ejecutaban el programa después de hacer varias correcciones, para ver los efectos del cambio. Los 3 expertos que introdujeron errores, los quitaron inmediatamente deshaciendo las modificaciones erroneas previas.

Las tablas 3.4.a, 3.4.b. y 3.4.c., muestran los tiempos acumulados de la corrección del error, para cada sujeto y cada error. Habia una diferencia considerable en el tiempo total de depuración entre las diferentes clases de sujetos. Los novatos, los de nivel medio y los expertos invertían en la tarea un promedio de 55,35 y 20 minutos respectivamente. Los novatos tardaron tanto en encontrar el error semántico, como los expertos tardaron en encontrar todos los errores. La mayoría de los novatos pasaban más de la mitad de su tiempo de depuración, buscando los errores lógicos.

Los novatos modificaron poco más o menos,casi el doble de sentencias (139) que los de nivel medio (62), mientras que los expertos sólo modificaron 53 sentencias. Cada error podía ser reparado cambiando sólo una sentencia como se muestra en el Apéndice A1. Los expertos cambiaron una única sentencia para reparar casi todos los errores. En muchos casos los novatos casi reescribieron totalmente la función o el procedimiento a base de ir haciendo cambios sucesivos. Mas de la mitad de los de nivel medio revisaron sustancialmente algunas funciones y procedimientos.

Tiempo acumulado para corrección error,en minutos

Nº del sujeto	N1	N2	N3	N4	N5	N6
Lectura inicial del programa	1	0	0	3	2	0
Para corrección de semántico1	14	8	19	5	16	10
Para corrección de semántico2	21	12	21	10	6	13
Para corrección de semántico3	22	16	*	*	20	25
Para corrección de lógico1	*	66	*	27	35	*
Para corrección de lógico2	*	55	*	*	6	*
Para corrección de lógico3	*	52	*	*	12	*
Total invertido en la depuración	67	66	55	55	35	58

Tabla 3.4a

(Aprendices)

* no pudo corregir ese error

Tiempo acumulado para corrección error,en minutos

Nº del sujeto	I1	I2	I3	I4	I5	I6
Lectura inicial del programa	3	0	2	4	0	0
Para corrección de semántico1	4	5	9	8	3	5
Para corrección de semántico2	6	16	12	10	5	14
Para corrección de semántico3	8	18	23	18	7	38
Para corrección de lógico1	5	22	38	32	31	43
Para corrección de lógico2	12	20	20	6	27	19
Para corrección de lógico3	17	*	28	17	25	*
Total invertido en la depuración	17	52	38	32	31	49

Tabla 3.4b

(Intermedios)

* no pudo corregir ese error

Tiempo acumulado para corrección error,en minutos

Nº del sujeto	E1	E2	E3	E4	E5	E6
Lectura inicial del programa	2	8	7	5	7	0
Para corrección de semántico1	14	12	11	8	17	3
Para corrección de semántico2	7	11	9	11	13	5
Para corrección de semántico3	11	11	12	20	13	7
Para corrección de lógico1	7	11	23	8	17	25
Para corrección de lógico2	7	11	9	11	13	25
Para corrección de lógico3	11	11	23	20	13	33
Total invertido en la depuración	14	12	23	20	17	33

Tabla 3.4c

(Expertos)

* no pudo corregir ese error

Los novatos y los de nivel medio se suponía que estaban algo familiarizados con las técnicas de clasificación por burbuja y por búsqueda binaria, pero en la realidad no parecía, que eran capaces de entender su funcionamiento muy correctamente.

A continuación mostramos algunos ejemplos, de las versiones del procedimiento LEERDATOS y del procedimiento CLASIFICACION y finalmente de la función de BUSQUEDA-binaria, construidas por novatos (figuras 3.1a, 3.1b y 3.1c). Todas ellas reflejan modificaciones extensas del procedimiento o de la función original.

Versión correcta de N4 del procedimiento LEERDATOS

```
procedure LEERDATOS(var ficent:text;a:tipoarray;
                    var contador:integer);

var indice:integer;

begin

  for indice:=1 to longitud do
      a[indice]:=0;

  indice:=1;

  while not eof(ficent) do

      begin

        while not eoln(ficent) do

          begin
            read(ficent,a[indice]);
            write(a[indice]);
            indice:=indice+1;
          end;

        readln(ficent);

      end;contador:=indice;
end.
```

Figura 3.1a

Versión correcta de N5 del procmiento. CLASIFICACION

```
procedure CLASIFICACION(var a:tipoarray;
                        contador:integer);
var
     i,j,temp:integer;
begin
     for i:=1 to contador -1 do
       for j:=i+1 to contador do
         if a[j]<a[i] then
           begin
             temp:=a[j];
             a[j]:=a[i];
             a[i]:=temp;
           end;
end;
```

Figura 3.1b

Versión correcta de N5 de BUSQUEDA-binaria

```
function BUSQUEDA(a:tipoarray;contador:integer;
                  clave:integer):integer;
var  lin,lsup,lmedio:integer;
begin
  lin:=1;
  lsup:=contador+1;
  lmedio:=trunc(lin+lsup)/2);
  while (lin<>lsup)and(a[lmedio]<>clave)do
    begin
    if clave<a[lmedio]then
          lsup:=lmedio
                       else
          lin:=lmedio+1;
    end;
    if clave=a[lmedio]then
          BUSQUEDA:=lmedio
    else
          BUSQUEDA:=0;
end;
```

Figura 3.1c

3.3 COMENTARIO

Los resultados de este estudio, proporcionan algunas interioridades de las diferencias entre programadores novatos,expertos y de nivel medio y el papel que juega la experiencia en la depuración. Como en estudios previos existentes sobre depuración, vemos que los expertos localizaron y corrigieron los errores más rápidamente que los novatos y que no introdujeron nuevos errores. De los datos obtenidos del método de análisis de protocolo , podemos empezar a deducir, las razones de estas diferencias que se estan comentando.

La estrategia de depuración utilizada, parecía ser la principal diferencia entre los expertos,los novatos y los de nivel medio. Parece ser que usaban 2 estrategias generales de depuración diferentes: Comprensión y Aislamiento.

Los expertos utilizaron una estrategia de comprensión, en la que primero alcanzaron un entendimiento global de lo que el programa tenia que hacer y de como funcionaba, según las sentencias escritas que lo componian y después usaban este conocimiento para localizar y corregir los errores que habia. Los novatos y los intermedios parecian intentar primero una estrategia de comprensión y luego se cambiaban a una estrategia de aislamiento en la que se centraban

enteramente, para corregir un error cada vez.

Quizás el apoyo más fuerte para la estrategia de comprensión de los expertos, era su habilidad para corregir múltiples errores a la vez,corregiendo errores semánticos y lógicos juntamente. A menudo estos errores estaban en diferentes procedimientos. Nuestros datos de protocolo mostraron que los expertos,hicieron estas correcciones de errores múltiples a ráfagasde tiempo. Este tipo de actividad, sugiere que los expertos, en su búsqueda por la causa de un error determinado, eran capaces de reconocer y corregir otros errores, con los que se iban encontrando. Parecería imposible hacer esto, sin una buena comprensión total del programa, porque esto implica reconocer un error en el contexto de un programa entero. Los expertos aparecían también bastante confiados en las correcciones que efectuaban. Ellos no paraban cada vez que hacían una corrección, ni tampoco la verificaban sino que lo que hacían,era esperar a que todas las correcciones estuvieran hechas,antes de procesar el programa para verificarlas.

Los datos para los errores semánticos y lógicos indican que los novatos e intermedios inicialmente intentaban usar una estrategia de comprensión y luego pasaban a la estrategia de aislamiento. Primero intentaban obtener un entendimiento global del programa, pero no lo conseguían porque adoptaban un seguimiento del orden de ejecución del programa erroneo, pues leían las sentencias

del programa en su orden físico. Debido a esta forma de acercamiento sólo eran capaces de alcanzar un entendimiento de bajo nivel de las distintas partes del programa. Esto está en marcado contraste con los expertos, que leían el programa en el mismo orden en el que se ejecutaba realmente y por tanto conseguían un entendimiento global completo. El fracaso de los programadores aprendices y de los de nivel medio, se podría explicar por qué dedicaban menos tiempo que los expertos, al estudio inicial del programa. Después de la lectura inicial, los aprendices y los de nivel medio se lanzaron rápidamente a probar el programa.

El programa contenía errores semánticos, por lo que la prueba terminaba dando mensajes de error semántico y el número de la sentencia donde el error ocurría. El mensaje de error y el indicador de la sentencia erronea, era lo que les incitaba para el cambio a una estrategia de aislamiento. Puesto que la mayoría de los de nivel medio y de los novatos encontraron todos los errores semánticos primero, esto sugiere que utilizaban con éxito en su estrategia de aislamiento, los mensajes de error y la información que recibían de los números de sentencia. Sin embargo, los errores lógicos eran más difíciles de encontrar utilizando la estrategia de aislamiento, porque la única información disponible eran las discrepancias existentes entre la salida real obtenida y la salida correcta esperada. No había información específica sobre el tipo de error o sobre dónde ocurría éste.

Esta falta de información específica del error, parece ser la explicación más probable a la pregunta de por qué los 4 novatos, que no pudieron encontrar todos los errores, entre todos ellos pudieron encontrar sólo un error lógico, sin embargo solo dejaron de encontrar 2 errores de los semánticos. Esto sugiere que los novatos necesitan información explícita, tal como mensajes de errores semánticos, para poder corregir la totalidad de ese tipo de errores.

Parecían incapaces de determinar qué procedimiento o función contenía los errores lógicos. Muchos pasaron bastante tiempo intentando corregir el error lógico en la función de BUSQUEDA-binaria, y no podían efectuar la corrección debida de los errores en los procedimientos LEERDATOS y CLASIFICACION.

La estrategia de aislamiento adoptada por los novatos y los de nivel medio, parecía explicar la razón por la qué sistemáticamente localizaban, corregían y verificaban un único error a la vez, empezando con los errores semánticos. Los datos de protocolo mostraron que cuando los novatos y los de nivel medio buscaron la causa de un error semántico generalmente limitaban su búsqueda a un procedimiento o función y frecuentemente pasaban por alto un error lógico en ese mismo procedimiento. Parecían estar buscando la causa para un error concreto y se olvidaban de los otros errores.

En un experimento sobre temas análogos Vessey, llegó a la conclusión de que los expertos y los novatos utilizaban la misma estrategia global de depuración, la comprensión. Pero hay que tener en cuenta, que todos sus sujetos eran programadores profesionales [Vess85]. Nuestros resultados en cambio sugieren la idea de que los expertos usan una estrategia de comprensión,mientras que los de nivel medio y los aprendices usan una estrategia de aislamiento. Creemos que algunos de nuestros programadores de nivel medio y novatos,probablemente han intentado una estrategia de comprensión al principio, antes de pasarse a la estrategia de aislamiento finalmente. Los resultados de Vessey pueden haber sido influenciados, por el hecho de que la tarea encomendada a sus sujetos no se ejecutaba en máquina,sino que se desarrollaba solo sobre papel.

Nuestros datos indicaban que los expertos estudiaban un programa el doble de tiempo, que los de nivel medio y los novatos,antes de hacer la primera modificación. Esto es inconsistente con el hallazgo de [Guge86], que decía que los novatos tardaban más tiempo estudiando el programa que los expertos. Creemos que los tipos de error en los programas cuentan mucho, si no todo, esta es la diferencia. Nuestro programa contenía errores semánticos y lógicos mientras que el programa de Gugerty contenía, un único error lógico. Nuestros sujetos recibieron un mensaje de error y un indicador de la sentencia donde el error ocurría, cuando procesaban el programa. Muchos de nuestros

sujetos lo que primero intentaron fué corregir este error. La única información disponible para los sujetos sobre el error lógico en el estudio de Gugerty,era la discrepancia existente entre el output real del programa y el output correcto. Debido a ello los sujetos pasaron el tiempo estudiando el listado del programa y utilizaron un block de notas, para la ejecución manual simulada del programa, para encontrar la causa del error lógico.

Otra diferencia era que los sujetos que intervenían en el experimento de Gugerty, conocían MacPascal y sólo se les dieron 25 minutos de explicaciones para saber Turbo-Pascal. Esto explicaría además por qué estos individuos no utilizaron herramientas de depuración ON-LINE, mientras que los nuestros si las utilizaron.

En todos los estudios previos sobre depuración,los sujetos hicieron poco uso de las ayudas de depuración ON-LINE. En nuestro estudio sólo los expertos hicieron uso de estas ayudas, para hacer trazas de variables. Sin embargo, nuestros novatos y los de nivel medio insertaron sentencias Write en la depuración. Muchas de estas, se localizaban dentro de bucles generando un considerable output y parecía claro que ellos sólo tenían una idea general sobre el error. Los pocos expertos que insertaron sentencias WRITE de depuración, parecían tener una idea clara acerca del error,ya que situaban las sentencias Write de la depuración al principio y al final de subprogramas y por ello

generaron poco output pero de gran valor. Esto parece demostrar la importancia de la experiencia y la práctica en la depuración.

3.4. CONCLUSIONES

Este estudio nos dió una idea del proceso de depuración que seguian los programadores estudiantes. La comprensión del programa, era la mayor diferencia existente entre las 3 clases de programadores estudiantes. Los expertos alcanzaron la mejor comprensión del programa, porque leyeron el programa en el orden en que realmente sería ejecutado; programa principal, procedimiento LEERDATOS, procedimiento CLASIFICAR y función de BUSQUEDA-binaria. Por otra parte, muchos de los de nivel medio y la mayoria de los novatos, leyeron el programa desde el principio hasta el final en orden puramente físico de escritura. Además los expertos eran capaces de usar un acercamiento de comprensión para depurar,mientras que los de nivel medio y los novatos utilizaron una aproximación al método de aislamiento.

El método de comprensión parece explicar, porqué los expertos eran mas rápidos a la hora de corregir los 6 errores y porqué a veces corregian varios errores a la vez.

El método de aislamiento parece explicar, por qué los de nivel medio y los novatos eran más lentos y corregían solo errores sueltos. La falta de la comprensión total del programa y de los mensajes de error explícitos, parecían explicar, por qué los novatos tenían tantos problemas corrigiendo los errores lógicos.

Los resultados de este estudio indicaron que la razón principal por la que los expertos tenían un mejor rendimiento a la hora de depurar errores era que comprendían mejor el programa. Sin embargo para poder afirmar que la comprensión del programa, juega un papel vital en el proceso de depuración del programa, necesitabamos medir la profundidad y el nivel de comprensión del programa por parte del sujeto después del estudio inicial del mismo, en el momoento de efectuar la primera modificación y al final de la sesión de depuración. Los 2 experimentos descritos en el capítulo siguiente tratarán esta cuestión.

CAPITULO 4

D I F E R E N C I A S E N T R E PROGRAMADORES EXPERTOS Y NOVATOS EN LA COMPRENSION DEL PROGRAMA EN LA DEPURACION

4.1. Introducción

En el estudio de protocolo, del capitulo 3 se comentó, que la diferencia principal existente entre los expertos y los aprendices, era que los expertos utilizaban una estrategia de comprensión en la depuración, mientras que los novatos y los de nivel medio usaban una estrategia de aislamiento, esto sugiere fuertemente, que la razón principal de que los expertos emplearan esta estrategia era su mejor entendimiento del programa. En otros estudios de protocolo [Guge86], [Kess86], [Vess85], la comprensión del programa por parte de los sujetos también parecía explicar, la mayoria de las diferencias de rendimiento total en la depuración entre expertos y novatos. Sin embargo, en los estudios mencionados y en nuestro estudio en particular,

el grado de comprensión del programa o si los sujetos alcanzaron la comprensión desde un estudio inicial del programa o si lo alcanzaron gradualmente durante el proceso, no estaba claro. Los dos estudios de protocolo de los que hablamos en este capitulo intentan aclarar estas cuestiones y establecer la hipotesis de que el grado de comprensión de un programa, es un buen predictor del rendimiento total de un programador en la depuración de un programa.

En ambos experimentos la comprensión del programa se media, al principio del proceso de depuración y de nuevo al finalizar la sesión de depuración. Se utilizó una tarea que consistia en reconstruir un programa para medir la comprensión inicial del mismo. En el primer experimento la tarea de reconstrucción era ejecutada inmediatamente seguido a efectuar la primera modificación por parte de los sujetos y era ejecutada después de 12 minutos en un segundo experimento. Para la medida de comprensión hecha al finalizar el experimento, se empleó una encuesta de comprensión en el primer experimento y un procedimiento de rellenar huecos en el segundo.

Una medida común para determinar el grado de comprensión del programa es la de realizar una tarea de reconstrucción de un programa. Sin embargo, la tarea de reconstrucción utilizada en este experimento es diferente de otras tareas similares, tratadas en la literatura

[Wied86], [Shne80], en 3 conceptos:

a.- Los sujetos reconstruian un programa defectuoso.

b.- A los sujetos no se les dijo antes del experimento que ellos tendrían que reconstruir un programa.

c.- Los sujetos conocian el funcionamiento global del programa defectuoso.

4.2. Experimento 2

4.2.1. Sujetos

Los sujetos de este experimento eran 6 novatos y 6 expertos, todos voluntarios. Los novatos habian terminado muy recientemente un curso introductorio de programación al PASCAL. El grupo de expertos estaba compuesto por estudiantes de informática de último curso. Todos los expertos eran programadores estudiantes muy experimentados.

4.2.2. Materiales

El programa a depurar era idéntico al usado en el

experimento 1, sólo que en éste, se práctica con un programa monolítico, más que con un programa modular. Puesto que la programación modular facilita la comprensión del programa por los sujetos, permitiéndoles concentrarse en un pequeño trozo de un programa y codificar ese trozo utilizando conceptos semánticos de más alto nivel, pensamos que un programa monolítico suministraría un mejor examen de comprensión.

El programa era un programa PASCAL de 50 líneas, con el que se leían 19 registros de datos enteros desde un fichero, se clasificaban los datos en un orden ascendente utilizando el algoritmo de clasificación de la burbuja y se buscaban 5 valores clave concretos en la lista clasificada usando un algoritmo de búsqueda binaria. Estaba escrito de una manera estructurada con nombres significativos e indentación, contenía 3 líneas de comentarios describiendo el programa, al inicio del mismo, pero no tenía comentarios mas abajo. Todos los sujetos estaban familiarizados con el algoritmo de búsqueda binaria y con el algoritmo de clasificación por el método de la burbuja utilizados en el programa.

Había 3 errores lógicos en el programa:

a.- Off-by-one error: el número de iteraciones a través de un bucle está desfasado en una.

b.- Error de sentencia de asignación: A una variable se le asigna un valor incorrecto.

c.- Error de predicado: Expresión condicional incorrecta.

Estos tres errores lógicos, eran idénticos a los usados en el experimento de protocolo descrito en el capitulo 3. Se seleccionaron estos errores porque son los que frecuentemente cometen los estudiantes y requieren una comprensión exhaustiva del dominio del problema. Los errores semánticos no se incluyeron en este experimento, porque nuestro estudio previo de protocolo indicó que los sujetos pueden reparar estos errores sin comprender el programa.

El listado del programa defectuoso, con los tres errores, se muestra en el Apéndice B1. Cada uno de estos errores podía ser corregido modificando sólo una sentencia para cada uno. Las versiones correctas de estas sentencias aparecen en el listado del programa defectuoso especificadas como comentarios.

4.2.3. Procedimiento

Los sujetos ejecutaron la tarea de depuración individualmente. Se les dió una copia impresa del programa

defectuoso y una copia impresa del fichero de datos de salida, ambos estaban también disponibles en un diskette, también se les dió una copia de la salida correcta y otra de la salida incorrecta (ver Apéndice B2).

A los sujetos no se les dijo cuántos o qué tipo de errores contenía el programa o que cada error podía ser reparado cambiando sólo una sentencia. Además, se les informaba a los sujetos de que ellos podían depurar el programa a su propio paso y usar cualquier técnica de depuración.

El procedimiento de recopilación de datos era idéntico al descrito en el capítulo anterior. Para medir la comprensión de los sujetos en el inicio del proceso de depuración, inmediatamente después de efectuar la primera modificación del programa, el programa y otros materiales les eran retirados y a los sujetos se les daba 20 minutos, para crear un programa que fuera lo mas parecido posible al original.

Además, se les mandaba que no cambiaran los algoritmos utilizados en el programa defectuoso. No se les dijo de antemano que se les iba a pedir una tarea de reconstrucción durante la sesión de depuración. Para la tarea de reconstrucción los sujetos empezaron con una plantilla del programa. La plantilla del programa se muestra en el Apéndice B3. Durante la tarea de reconstrucción, se grabó

el orden en que las sentencias del programa se reconstruían, el tiempo empleado en cada una de las tres rutinas del programa, y el número total de referencias a la parte de declaración contenida en la plantilla del programa. Después de la tarea de reconstrucción del programa, los sujetos reanudaron su sesión de depuración.

Finalmente, después de la sesión de depuración, a los sujetos se les dió la versión correcta del programa y se les pidió que respondieran a 9 preguntas de un examen de comprensión acerca del programa. El test se muestra en el Apéndice B4. El propósito de este test era medir la comprensión del programa de los sujetos a la finalización de la sesión de depuración.

4.2.4. Resultados

Aunque el programa defectuoso se ejecutaba como un programa monolítico, contenía 3 funciones distintas: leer 19 registros de datos enteros del fichero, clasificar todas las unidades de datos enteros y buscar 5 valores clave en la lista clasificada. La nota para el programa reconstruido, estaba basada en las notas que se daban para cada parte del programa. Se utilizaban dos maneras de calificación, una basada en el funcionamiento correcto del programa y el otro basado en la reconstrucción del programa palabra por palabra.

Para la calificación del funcionamiento las tres funciones se descomponían en un número de módulos. Cada módulo consistía en una o más sentencias del programa y representaba un modelo conocido en el algoritmo especificado. Las diferentes partes del programa para cada una de estas funciones consistían en lo siguiente:

Para leer los datos:

a.- Inicializar los índices o las variables contador.

b.- Iterar para leer una serie de valores enteros.

c.- Leer un valor entero y asignar su valor a un elemento matriz.

d.- Incrementar la variable índice en uno.

e.- Contar el número total de valores de entrada.

Para clasificar los datos por el método de la burbuja:

a.- Iterar tantas veces como indique el contador.

b.- Iterar para comparar los valores de la matriz en cada iteración.

c.- Comparar dos elementos de matriz sucesivos.

d.- Intercambiar dos elementos de matriz si es necesario.

Para la búsqueda de los valores clave mediante el método de búsqueda binaria:

a.- Recibir el nº total de valores clave.

b.- Leer un valor clave.

c.- Inicializar el principio y el final de los elementos de matriz.

d.- Iterar el proceso de búsqueda.

e.- Calcular la posición del elemento situado en la mitad de la matriz.

f.- Probar la igualdad y establecer los valores de índices adecuados.

g.- Terminar el proceso de búsqueda.

h.- Volver a hacer todo el proceso hasta terminar con todos los valores clave pedidos.

La puntuación total de la reconstrucción funcional (puntuación de todos los procesos anteriores), se basaba en el número total de las partes del programa que los sujetos reconstruyeron correctamente.

La tabla 4.1. muestra el número total de partes reconstruidas, el número de partes reconstruidas correctamente, y el número de partes que faltaban en la reconstrucción por trozos y para ambos, novatos y expertos. Si un sujeto reconstruía un algoritmo diferente, (por ejemplo búsqueda lineal en lugar de búsqueda binaria) se consideraba que tenía falta faltaba en todas las partes de ese algoritmo. La puntuación funcional para cada sujeto se muestra también en la tabla 4.2.

Como se puede ver en la tabla 4.1. y en la tabla 4.2., había diferencias considerables entre los programadores expertos y novatos. Los expertos reconstruyeron más partes (promedio = 15,5 para los expertos, y promedio = 11,8 para los aprendices), más partes correctamente (promedio = 13,5 para los expertos, promedio = 8,5 para los aprendices) y tenían menos partes omitidas (promedio = 0,5 para los

expertos, y promedio = 4,2 para aprendices) que los aprendices. Para cada uno de los 3 parámetros, se obtuvieron diferencias estadísticas muy significativas entre programadores expertos y aprendices. Para cada uno de los 3 segmentos del programa, los expertos reconstruyeron más partes que los novatos y con un porcentaje mayor de partes reconstruidas correctamente. También los expertos tenian menos partes omitidas para cada segmento. Tanto los expertos como los novatos reconstruyeron mejor las instrucciones contenidas en el segmento escrito para leer los datos y las contenidas en el de clasificarlos, que las contenidas en el segmento que se encargaba de efectuar la búsqueda de los valores clave. Los expertos reconstruyeron casi el doble número de partes de la búsqueda binaria que los novatos, además los novatos tenían un promedio de 3 partes omitidas en la búsqueda binaria, mientras que sólo uno de los expertos omitió una parte.

Para la calificación de la reconstrucción palabra por palabra, se consideraba como correcta una sentencia, si ésta era idéntica al original, excepto la indentación y espaciado. Se contaba como incorrecto, si los sujetos utilizaban diferentes nombres de variables, si cambiaban el orden de sentencias o si omitian una o más sentencias.

Todos los comentarios internos dentro de las ejecutables en la reconstrucción fueron ignorados. Además,

si un sujeto reconstruía correctamente una sentencia incorrecta (por ejemplo contador : = indice) del programa defectuoso, se consideraba incorrecta. Se daba un punto para cada sentencia reconstruida correctamente. Una puntuación palabra por palabra del sujeto, consistía en la suma de los puntos obtenidos mediante lo anteriormente comentado.

La tabla 4.3. muestra el número total de las sentencias del programa reconstruidas palabra por palabra por los sujetos. Como se puede ver, los programadores expertos reconstruyeron más sentencias del programa palabra por palabra que los programadores novatos (porcentaje medio = 40,9% para los aprendices, porcentaje medio = 50,9% para los expertos).

La mayor parte de esta diferencia se reflejaba, en la reconstrucción del segmento de la búsqueda binaria. Es importante anotar que los expertos y los novatos reconstruyeron los pares principio-fin, mejor que cualquier otra sentencia del programa.

La puntuación palabra por palabra de cada sujeto se muestra en la tabla 4.4. Sorprendentemente los novatos tenían mayor puntuación palabra por palabra, que los expertos en el segmento correspondiente a la lectura de los datos (promedio = 4,3 para novatos y promedio 3,5 para expertos), pero ellos puntuaron menos que los expertos en

el segmento que se ocupaba de clasificar los datos (promedio = 3,3 para novatos y promedio = 4,3 para expertos) y en el que se ocupaba de efectuar las búsquedas pedidas (promedio = 6,7 para novatos y promedio = 10,0 para expertos). Se advierte que la puntuación de los expertos es substancialmente mayor que la de los novatos, en el segmento que realiza la búsqueda mediante el algoritmo de búsqueda-binaria.

Puntuación funcional por partes del programa

	Novatos			Intermedios		
Partes del Programa	Partes re-cons truidas	Re-cons truidas bien	Partes omiti-das	Partes re-cons truidas	Re-cons truidas bien	Partes omiti-das
L partel	5	4	1	6	6	0
E parte2	6	5	0	6	6	0
E parte3	6	4	0	6	6	0
R parte4	6	6	0	6	6	0
parte5	2	1	4	4	3	2
%	83%	67%	17%	93%	90%	7%
C partel	6	3	0	6	6	0
L parte2	5	2	1	6	5	0
A parte3	6	4	0	6	5	0
S parte4	6	6	0	6	6	0
I %	96%	63%	4%	100%	92%	0%
B partel	2	2	4	5	5	1
U parte2	4	4	2	6	6	0
S parte3	4	2	2	6	6	0
Q parte4	4	3	2	6	4	0
U parte5	3	4	3	6	5	0
E parte6	3	0	3	6	2	0
D parte7	3	1	3	6	4	0
A %	55%	38%	45%	98%	76%	2%
total	71	51	25	93	81	3
%global	74%	53%	26%	97%	84%	3%

Tabla 4.1

Puntuación funcional de novatos y expertos

Sujetos	Nº partes re-construidas	Nºpartes recons truidas bien	Nº partes omitidas
Novatos			
N1	16	12	0
N2	9	6	7
N3	16	10	0
N4	6	5	10
N5	9	5	7
N6	15	13	1
total	71	51	25
media	11.8	8.5	4.2
%	74%	53%	26%
Expertos			
E1	16	13	0
E2	15	15	1
E3	15	10	1
E4	16	14	0
E5	16	14	0
E6	15	15	1
total	93	81	3
media	15.5	13.5	0.5
%	97%	84%	3%

Tabla 4.2

Nº total de instrucciones rehechas palabra por palabra

Instrucciones de programa	Novat	Exper
indice:=1;	3	0
while not eof(ficent) do	3	4
begin	6	6
readln(ficent,a[indice]);	3	2
indice:=indice+1;	4	2
end;	6	6
contador:=indice;	1	1
for i:=1 to contador -1 do	2	2
for j:=1 to contador-1 do	1	2
if a[j]>a[j+1]then	2	4
begin	6	6
temp:=a[j];	3	6
a[j+1]:=a[j];	0	0
a[j]:=temp;	0	0
end;	6	6
for i:=1 to numcla do	1	2
begin	3	5
write('clave=');	4	2
readln(clave);	4	6
lin:=1;	3	6
lsup:=contador;	2	2
while lin<>lsup do	1	2
begin	3	6
lmedio:=(lin+lsup)div2;	2	5
if clave>=a[lmedio]then	1	3
lsup:=lmedio	2	1
else	3	5
lin:=lmedio+1;	2	2
end;	3	6
if clave=a[lin]then	0	1
lugar:=lin;	0	0
else	1	1
lugar:=0;	1	0
writeln('clave=',clave,'valor=',lugar);	1	0
end;	3	5
total	86	107
media	14.3	17.8
porcentaje	40.9%	50.9%

Tabla 4.3

Puntuación palabra por palabra de novatos y expertos

Sujetos	Nombre del procedimiento o de la función			
	LEERDATOS	CLASIFICACION	BUSQUEDA	total
Novatos				
N1	6	5	13	24
N2	5	3	2	10
N3	4	4	10	18
N4	3	3	2	8
N5	3	0	3	6
N6	5	5	10	20
total	26	20	40	86
media	4.3	3.3	6.7	14.3
Expertos				
E1	4	4	8	16
E2	3	4	11	18
E3	3	4	4	11
E4	6	5	12	23
E5	0	3	11	14
E6	5	6	14	25
total	21	26	60	107
media	3.5	4.3	10.0	17.8

Tabla 4.4

Expertos y novatos pasaron casi el mismo tiempo (aproximadamente 6 minutos), antes de su tarea de reconstruir y excepto para desarrollar el segmento de la búsqueda, reconstruyeron casi el mismo número de sentencias del programa. Además, habia 3 pequeñas diferencias en el tiempo global medio para la reconstrucción entre los expertos y los novatos.

El propósito del examen de comprensión posterior a la sesión, era cuantificar la comprensión del programa por parte de los sujetos al final de la depuración. Habia 9 preguntas en el examen de comprensión, 5 de las cuales tenían 2 partes, y se daba un punto para cada respuesta correcta. Esto daba una puntuación máxima en el examen de comprensión de 14 puntos. La puntuación del examen de comprensión de los sujetos del programa después de la sesión, junto con una puntuación en la reconstrución palabra por palabra, la puntuación funcional (el número total de partes reconstruidas correctamente) y el número de errores corregidos, aparecen en la tabla 4.5., como se puede ver en esta tabla los novatos puntuaron más bajo que los expertos (promedio = 10,8 para los novatos y promedio = 12,8 para los expertos) en el examen de comprensión. Sin embargo, esta diferencia no era muy significativa.

Puntuación para la comprensión en la depuración

Sujetos	Puntuación palabra por palabra	Puntuación funcional	Puntuación del exámen después de la depuración	Número de errores corregidos
Novatos				
N1	24	12	14	3
N2	10	6	7	2
N3	18	10	10	1
N4	8	5	11	1
N5	6	5	8	1
N6	20	13	14	3
total	86	51	64	11
media	14.3	8.5	10.7	1.8
Expertos				
E1	16	13	14	3
E2	18	15	14	3
E3	11	10	10	2
E4	23	14	12	3
E5	14	14	13	3
E6	25	15	14	3
total	107	81	77	17
media	17.8	13.5	12.8	2.8

Tabla 4.5

Hay que advertir que dos novatos (N1 y N6) que corrigieron los 3 errores, tenían las dos reconstrucciones palabra a palabra más perfectas, así como la reconstrucción de los subprogramas y la mayor puntuación del examen después de la sesión, de entre los novatos.

Hay que advertir también, que el experto (E3) que falló en corregir los 3 errores, tuviera la reconstrucción peor y las puntuaciones del examen posterior a la sesión entre los expertos mas bajas. El coeficiente de correlación se calculó entre el rendimiento total de los sujetos en la depuración (número total de errores corregidos) y el rendimiento total de la reconstrucción del programa (puntuación funcional, puntuación palabra por palabra y la puntuación de la comprensión posterior ($r = 0,86$, $r = 0,70$, y $r = 0,77$, respectivamente). Estos coeficientes de correlación relativamente altos parecen indicar, que la comprensión de los sujetos del programa está relacionada con su rendimiento total en la depuración.

El orden de corrección del error de los expertos y los novatos se muestra en la tabla 4.6. Los cuatro aprendices (N2, N3, N4, N5) y un experto (E3), que no consiguieron corregir todos los errores, ninguno de ellos logró encontrar el error en el segmento de búsqueda-binaria. Hay que advertir que todos los sujetos excepto E2, localizaron y corrigieron un único error cada vez.

Orden de corrección del error

Sujetos	Error lógico en LEERDATOS	Error lógico en CLASIFICACION	Error lógico en BUSQUEDA-binaria
Novatos			
N1	1	2	3
N2	2	1	*
N3	*	1	*
N4	1	*	*
N5	*	1	*
N6	3	2	1
Expertos			
E1	2	3	1
E2	2	2	1
E3	2	1	*
E4	1	2	3
E5	2	1	3
E6	1	2	3

Tabla 4.6

* no corrigió el error

4.3. Experimento 3

4.3.1. Sujetos

En este experimento participó otro grupo de programadores diferente al del anterior experimento.

Habia 6 sujetos en cada grupo. Al igual que en el experimento anterior, los novatos estaban terminando un curso de programación introductorio al PASCAL y el grupo de expertos, estaba compuesto de estudiantes de último curso de Informática.

4.3.2. Materiales

El programa utilizado era idéntico al que se usó en el anterior experimento. Ver apéndice B1.

4.3.3. Procedimiento

Como en el experimento anterior, a los sujetos se les presentaron listados del programa defectuoso, archivo de los datos de entrada, la salida incorrecta, la salida correcta y el programa en un diskette. A los sujetos se

les comentó que depuraran el programa, a su propio paso y que usaran cualquier técnica de depuración que conocieran.

Sin embargo, no se les dijo el número total de errores del programa ni el tipo de errores que contenía el programa. Durante la sesión de depuración, un ayudante del investigador grabó las actividades finales de los sujetos. Después de 12 minutos desde el comienzo de la depuración del programa, se recogieron todos los materiales y se les encargó una tarea de reconstrucción de un programa que duraba 20 minutos en total. Después de esta tarea de reconstrucción del programa, los sujetos reanudaron la sesión de depuración.

Finalmente, después de la sesión de depuración, se les dió a los sujetos, un test de comprensión, que consistía en el procedimiento de rellenar huecos. Este tipo de test sigue el procedimiento de "rellenar las partes omitidas". Algunas partes del programa se suprimieron sistemáticamente y se reemplazaron por espacios en blanco. La habilidad de un sujeto para rellenar los huecos, se relaciona con el grado de entendimiento que tiene del programa. La versión del procedimiento de rellenar huecos en el programa, se muestra en el Apéndice B5.

4.3.4. Resultados

Puesto que este experimento es casi idéntico al experimento anterior, la reconstrucción de los programas por parte de los sujetos fué calificada,utilizando los esquemas de calificación del experimento anterior. Aunque la tarea de reconstruir, fué encargada al transcurrir 12 minutos desde el inicio de la depuración, casi todos los resultados eran los mismos o similares a los encontrados en el experimento anterior. El resto de esta sección describe brevemente estos resultados.

Los puntos por la reconstrucción funcional de todos los sujetos se muestran en las tablas 4.7. y 4.8.. De nuevo, los expertos reconstruyeron más partes de cada una de las tres funciones a realizar, que los novatos (promedio = 13,5 para los aprendices y promedio = 15,8 para los expertos) y las reconstruyeron correctamente más a menudo (promedio = 10,3 para los novatos y promedio = 13,1 para los expertos). Las diferencias eran significativas entre expertos y aprendices. Además, los expertos eran mucho mejores que los novatos en reconstruir el segmento de la búsqueda-binaria. También, hay que advertir que todos los expertos excepto El reconstruyeron todas las partes del programa, mientras que de los novatos (N4 y N6) solo dos lo consiguieron. La reconstrucción de los sujetos fue mejor que la de los del experimento 2, porque tenían más tiempo para estudiar y trabajar con el programa.

Puntuación funcional por partes del programa

Partes del Programa	Novatos			Intermedios		
	Partes re-construidas	Re-construidas bien	Partes omitidas	Partes re-construidas	Re-construidas bien	Partes omitidas
L parte1	6	6	0	6	6	0
E parte2	6	5	0	6	6	0
E parte3	6	5	0	6	5	0
R parte4	6	5	0	6	6	0
parte5	4	2	2	5	3	1
%	93%	77%	7%	97%	87%	3%
C parte1	5	5	1	6	6	0
L parte2	6	2	0	6	4	0
A parte3	6	3	0	6	6	0
S parte4	6	6	0	6	6	0
I %	96%	67%	4%	100%	92%	0%
B parte1	5	5	1	6	6	0
U parte2	5	5	1	6	6	0
S parte3	4	3	2	6	5	0
O parte4	3	2	3	6	2	0
U parte5	5	4	1	6	6	0
E parte6	4	2	2	6	4	0
D parte7	4	2	2	6	4	0
A %	71%	55%	29%	100%	74%	0%
total	81	62	15	95	79	1
%global	84%	65%	16%	99%	82%	1%

Tabla 4.7

Puntuación funcional de novatos y expertos

Sujetos	Nº partes reconstruidas	Nº partes rehechas bien	Nº partes omitidas
Novatos			
N1	14	9	2
N2	14	9	2
N3	11	10	5
N4	16	14	0
N5	10	6	6
N6	16	14	0
total	81	62	15
media	13.5	10.3	2.5
%	84%	65%	16%
Expertos			
E1	15	11	1
E2	16	13	0
E3	16	15	0
E4	16	13	0
E5	16	15	0
E6	16	12	0
total	95	79	1
media	15.8	13.2	0.2
%	99%	82%	1%

Tabla 4.8

Las tablas 4.9. y 4.10. muestran el número total de sentencias, reconstruidas palabra por palabra por los sujetos. Los programadores expertos reconstruyeron más sentencias del programa palabra por palabra que los programadores novatos (media en tanto por ciento : 48,5 para los aprendices y 59,0% para los expertos). Como en el experimento 2, los programadores expertos reconstruyeron las formas sintácticas, mejor que otras sentencias del programa y reconstruyeron casi el mismo número de sentencias (promedio = 32 para los novatos y promedio = 34 para los expertos) en total.

Nº total de instrucciones rehechas palabra por palabra

Instrucciones de programa	Novat	Exper
indice:=1;	3	1
while not eof(ficent) do	5	5
begin	6	6
readln(ficent,a[indice]);	4	4
indice:=indice+1;	4	4
end;	6	6
contador:=indice;	2	1
for i:=1 to contador -1 do	3	4
for j:=1 to contador -1 do	2	4
if a[j]>a[j+1]then	2	4
begin	6	6
temp:=a[j];	6	6
a[j+1]:=a[j];	0	0
a[j]:=temp;	0	0
end;	6	6
for i:=1 to numcla do	2	2
begin	4	6
write('clave=');	4	1
readln(clave);	3	6
lin:=1;	4	6
lsup:=contador;	4	3
while lin<>lsup do	1	3
begin	3	6
lmedio:=(lin+lsup)div2;	4	6
if clave>=a[lmedio]then	2	2
lsup:=lmedio	1	3
else	3	3
lin:=lmedio+1;	1	3
end;	3	6
if clave=a[lin]then	2	2
lugar:=lin;	0	0
else	2	3
lugar:=0;	0	0
writeln('clave=',clave,'valor=',lugar);	0	0
end;	4	6
total	102	124
media	17.0	20.7
porcentaje	48.5%	59.0%

Tabla 4.9

Puntuación palabra por palabra de novatos y expertos

Sujetos	Nombre del procedimiento o de la función			
	LEERDATOS	CLASIFICACION	BUSQUEDA	total
Novatos				
N1	4	4	3	11
N2	5	4	7	16
N3	4	4	4	12
N4	7	6	15	28
N5	4	2	3	9
N6	6	5	15	26
total	30	25	47	102
media	5.0	4.2	7.8	17.0
Expertos				
E1	3	4	6	13
E2	7	4	14	25
E3	4	6	13	23
E4	7	6	14	27
E5	3	6	10	19
E6	3	4	10	17
total	27	30	67	124
media	4.5	5.0	11.2	20.7

Tabla 4.10

La puntuación de la comprensión del programa obtenida por los sujetos al finalizar la sesión, se muestra en la tabla 4.11. Los novatos puntuaron menos en su tarea de comprensión que los expertos (promedio = 8,8 para los novatos, y promedio = 9,2 para los expertos) y tardaron más tiempo en completar la tarea de rellenar huecos, al final de la sesión, que los expertos (promedio = 8,3 minutos para los novatos, y promedio 6,2 minutos para los expertos).. El análisis de estos datos no marcó diferencias demasiado significativas entre expertos y novatos. Hay que advertir que los dos novatos (N4 y N6) que corrigieron todos los errores, tuvieron la puntuación mas alta en la reconstrucción palabra a palabra y en la sesión final, de entre los novatos. El experto (E6) que no corrigió todos los errores, tenía la puntuación de sesión final más baja de entre los expertos.

Como en el experimento 2, se observó que había una relación muy directa entre el rendimiento total en la depuración de los sujetos (número total de errores corregidos) y el rendimiento total en la reconstrucción del programa (puntuación funcional, puntuación palabra por palabra y la puntuación de la comprensión final ($r = 0.77$, $r = 0.76$ y $r = 0.64$ respectivamente).

Puntuación para la comprensión en la depuración

Sujetos	Puntuación palabra por palabra	Puntuación funcional	Puntuación del exámen después de la depuración	Número de errores corregidos
Novatos				
N1	11	9	8	1
N2	16	9	9	2
N3	12	10	8	0
N4	28	14	11	3
N5	9	6	7	1
N6	26	14	10	3
total	102	62	53	10
media	17.0	10.3	8.8	1.7
Expertos				
E1	13	11	11	3
E2	25	13	9	3
E3	23	15	11	3
E4	27	13	8	3
E5	19	15	9	3
E6	17	12	7	2
total	124	79	55	17
media	20.7	13.2	9.2	2.8

Tabla 4.11

El orden de corrección del error de los sujetos se muestra en la tabla 4.12. Todos los expertos excepto el E6, localizaron y corrigieron todos los errores con éxito, mientras que sólo dos novatos (N4 y N6) corrigieron todos los errores. Los expertos emplearon más tiempo en leer el programa, antes de realizar la primera modificación del programa (promedio = 4,8 minutos para los novatos, y promedio = 7,7 minutos para los expertos). El orden de corrección de los sujetos era similar al encontrado en experimentos anteriores.

Los expertos eran más eficaces que los novatos en la depuración del programa (media del tiempo de depuración = 35,0 minutos para los novatos, y media del tiempo de depuración = 29,2 minutos para los expertos). Puesto que otras medidas del rendimiento total de la depuración, estrategias de depuración y actividades fueron similares a las encontradas en el experimento 2, no se exponen en esta sesión.

Orden de corrección del error

Sujetos	Error lógico en LEERDATOS	Error lógico en CLASIFICACION	Error lógico en BUSQUEDA-binaria
Novatos			
N1	*	*	1
N2	*	1	2
N3	*	*	*
N4	1	2	3
N5	*	1	*
N6	3	1	2
Expertos			
E1	3	1	2
E2	3	1	2
E3	3	1	2
E4	1	2	3
E5	1	1	2
E6	2	1	*

Tabla 4.12

* no corrigió el error

4.4. Discusión

En los experimentos 2 y 3 la comprensión del programa por parte de los sujetos, se midió durante y después de la sesión de depuración. Los resultados indicaron una fuerte conexión entre las puntuaciones de la comprensión del programa por parte de los sujetos y el número de errores corregidos. Esto confirma la importancia de la comprensión del programa sugerida en los estudios previos sobre depuración. Los resultados también duplicaron muchos de los resultados obtenidos en el estudio de protocolo expuesto en el Capítulo 3. Por ejemplo, los programadores expertos eran más eficaces en localizar y corregir errores; Los expertos corregían casi todos los errores y además lo hacían mucho más rápido que los novatos.

Sin embargo, la estrategia de corrección de errores múltiples expuesta en el experimento 1 no fué respaldada por estos datos experimentales. Una explicación lógica para este comportamiento puede ser debido al hecho de que sólo se insertaron 3 errores lógicos y no se insertaron errores semánticos en el programa.

Un resultado sorprendente fué que todos los sujetos reconstruyeron la versión correcta del error de asignación incorrecto, en la parte correspondiente de permutaciones de valores de la rutina clasificación. Unos pocos expertos

también reconstruyeron el error off-by-one, en el segmento de la lectura de los datos, correctamente. Se debería tener en cuenta que ninguno de los expertos o novatos incluyeron la versión correcta del error de predicado en la reconstrucción de la búsqueda-binaria. Esto parece indicar que los expertos y los novatos trocearon las sentencias relacionadas en común tales como las que efectuaban la permutación por ejemplo y las memorizan como una unidad funcional, más que tratar cada sentencia como una unidad independiente para que se recuerden separadamente. Además no parecía que se dieran cuenta, de que estas sentencias eran incorrectas en el programa, porque casi todos los sujetos no pudieron localizar y corregir estos errores de asignación inmediatamente después de su tarea de reconstrucción.

No tenemos una buena explicación para su comportamiento, como no sea la de que los sujetos parecieron asumir que las relativamente difíciles secciones del código eran las que contenían el error. Todos los sujetos de ambos experimentos pensaron inicialmente que la parte del programa correspondiente a la búsqueda-binaria, era enteramente responsable del comportamiento anómalo del programa y modificaron una o más sentencias de este grupo funcional.

Programadores expertos y novatos reconstruyeron casi el mismo número de sentencias en la tarea de reconstrucción del programa. Este resultado es inconsistente con gran diferencia del expuesto por [Wied86], [Shne77], [Myer79]. En los estudios anteriores, el programa presentado a los sujetos no contenia comentarios describiendo la función global del programa y no se les suministró un programa patrón conteniendo las declaraciones para la tarea de reconstruir. Nuestros expertos tuvieron más éxito que los novatos en la reconstrucción del programa entero. Los novatos reconstruyeron a menudo sentencias redundantes tales como pares empezar-acabar, sentencias incorrectas y sentencias que no eran relevantes para el algoritmo utilizado en el programa. Por ejemplo, incluso aunque se les hubiera dicho que reconstruyeran el programa original, 2 novatos del primer experimento reemplazaron la búsqueda binaria por la búsqueda lineal en la reconstrucción.

En ambos experimentos, los sujetos usaron un programa patrón, suministrado por el investigador para reconstruir el programa original defectuoso. Como se esperaba, en estos experimentos, la puntuación de la reconstrucción palabra por palabra de los programadores expertos, era superior a la de los programadores novatos. Conviene anotar las siguientes 3 observaciones sobre su reconstrucción del programa:

a.- Los programadores expertos usaron nombres de variables más apropiadamente, que los programadores novatos. Por ejemplo, casi todos los expertos usaron el nombre variable "indice" como una variable de índice, mientras que los novatos la usaron como una variable para almacenar el número de elementos de los datos leídos del archivo de entrada.

b.- Los novatos, frecuentemente, añadían un nuevo nombre de variable en el programa e intentaban usar esta variable, en el segmento de búsqueda-binaria. Los novatos, que intentaron reconstruir la BUSQUEDA-binaria, reconstruyeron la versión de esa rutina, vista en un libro concreto que leyeron en el cursillo del lenguaje. Como resultado de esa reconstrucción obtuvieron una versión diferente, de la existente en el programa utilizado para la prueba.

c.- Ninguno de los programadores expertos intentó reconstruir un algoritmo diferente de los usados en el programa defectuoso, mientras que algunos de los novatos reconstruyeron un algoritmo de búsqueda diferente al del algoritmo de búsqueda-binaria.

Estas explicaciones pueden aclararnos en parte por qué los programadores expertos eran mas eficientes que los novatos, en reconstruir sentencias del programa.

Los programadores expertos y novatos del experimento 2 obtuvieron una puntuación de reconstrucción palabra por palabra, más alta que los del experimento 1 (ver tabla 4.13.). Gran parte de esa diferencia se puede atribuir al mayor tiempo invertido en la asimilación del programa, antes de la tarea de reconstrucción. Como se mencionó en la sección donde se explicaba el desarrollo de los experimentos, los sujetos del experimento 2 participaron en la tarea de la reconstrucción después de efectuar su 1ª modificación en el programa, mientras que los sujetos del experimento 3 reconstruyeron el programa después de llevar 12 minutos de sesión de depuración del programa. Esta diferencia de procedimiento permitió a los sujetos del experimento 3 modificar, por termino medio, más sentencias del programa que los sujetos del experimento 2.

Todos los sujetos de ambos experimentos intentaron comprender inicialmente el programa, leyendo el listado del programa defectuoso, los datos de entrada, la salida esperada y la salida real. Los programadores expertos emplearon más tiempo en la lectura inicial del programa, que los novatos. Esta observación si es consistente con los resultados obtenidos en el capítulo 3. En el estudio de protocolo anterior, sin embargo, 3 novatos, 3 de nivel medio, y un experto no intentaron entender el programa por medio de una lectura inicial e inmediatamente ejecutaron el programa. Esto puede tener su explicación, por el tipo de errores de que se trataba.

Se habían insertado errores semánticos y lógicos en el programa utilizado para el primer experimento, mientras que solo se insertaron errores lógicos para los experimentos expuestos en este capítulo. Así pues parecía que el mensaje de error semántico, junto con el número de línea del programa donde el error ocurría, incitaba a los novatos y a los de nivel medio a ejecutar el programa inmediatamente sin intentar entenderlo, al comienzo de la sesión de depuración.

Esto implica que los novatos y los de nivel medio, tienden a usar esta información explícita del error para aislar el error a un segmento concreto del programa, con el propósito de limitar su espacio de búsqueda y emplear estrategias sensitivas con los errores, para localizarlos y corregirlos.

Comparación entre el experimento 2 y el experimento 3

Capacidad de los sujetos en porcentaje	Experimento 2	Experimento 3
Novatos		
Puntuación palabra a palabra	41 %	49 %
Puntuación funcional	53 %	65 %
Puntuación del exámen después de la depuración	76 %	80 %
Número de errores corregidos	61 %	56 %
Expertos		
Puntuación palabra a palabra	51 %	59 %
Puntuación funcional	84 %	82 %
Puntuación del exámen después de la depuración	92 %	83 %
Número de errores corregidos	94 %	94 %

Tabla 4.13

En la tarea de reconstrucción que se ejecutó durante 20 minutos en los experimentos 2 y 3, el investigador reunió información sobre el orden en el que los sujetos reconstruyeron las sentencias del programa una a una.

Por lo general la reconstrucción por parte de los expertos, fué mas fluida que la de los novatos ya que estos frecuentemente modificaban sentencias del programa.

Normalmente, todos los expertos reconstruyeron un grupo de sentencias, se detuvieron unos pocos segundos y luego reanudaron la reconstrucción. Parece indicar que los expertos cogieron un grupo de sentencias del programa y las reconstruyeron como una única unidad. El troceo del programa por parte de los expertos, era bastante evidente según sus comentarios interlíneas en el programa reconstruído. Por ejemplo, los expertos E2, E5 y E6 del experimento 2, y E4 en el experimento 3, insertaron o una línea en blanco o un comentario interlínea al principio de cada unidad funcional, mientras reconstruían el programa. Durante la reconstrucción ninguno de los novatos insertó ninguna línea en blanco o comentarios entre líneas, entre cada función del programa. Esta observación, mas la modificación de los novatos, de las sentencias previamente reconstruídas, parece indicar que no agruparon las sentencias del programa tanto como los expertos.

Se anotaron varias medidas de depuración tales como el tiempo de lectura inicial del programa, el tiempo total de depuración y el número total de errores corregidos, para todos los expertos y para todos los novatos. Todos emplearon casi la misma cantidad de tiempo en leer el programa antes de su 1ª modificación del programa. Ninguno de los novatos o expertos procesaron el programa inmediatamente, sin haber hecho antes ninguna lectura inicial del programa. Incluso aunque 2 novatos (N2 y N5) emplearon relativamente más tiempo que los otros en la lectura inicial, no pudieron corregir todos los errores del programa. Todas las demás medidas de rendimiento total en la depuración, número de ejecuciones, número de modificaciones y número de errores introducidos de los expertos y novatos fueron las mismas o similares a las encontradas en el experimento 1. Esto parece indicar que los principales resultados obtenidos en el estudio anterior son similares.

Durante la sesión de depuración, los sujetos ejecutaron varias actividades. Por ejemplo, al principio de la depuración, todos los sujetos hicieron un primer exámen del listado de datos de entrada, del output real, del output esperado, y del listado del programa defectuoso, para encontrar una pista sobre el error y aislar el error a la función de búsqueda binaria. Ninguno de los sujetos sin embargo, ejecutó el programa inmediatamente sin una lectura inicial de él. Ya que el programa defectuoso

estaba implementado como un programa monolítico, todos los sujetos leyeron el programa desde el principio hasta el final en un orden físico. Con el fin de rastrear el valor de una cierta variable, los novatos insertaron muchas sentencias dentro de los loops, mientras que casi todos los expertos usaron las ayudas de depuración ON-LINE. Ambos, expertos y novatos, ejecutaron actividades idénticas durante la depuración, pero los expertos parece que las realizaron más eficazmente y con más éxito que los novatos.

4.5. Conclusiones

Se puede deducir de los estudios de depuración anteriores, que compararon a los novatos y expertos, que la razón principal del rendimiento total superior de los expertos en la depuración, era una comprensión mejor del programa. El experimento 1 sugirió esto firmemente. Los resultados de los experimentos 2 y 3 proporcionaron la 1ª evidencia concluyente de que los expertos obtenían una comprensión mejor del programa.

La comprensión del programa, puede que no sea tan importante para localizar y corregir errores semánticos, porque el mensaje de error con la información del número de línea a menudo ayuda a los programadores a identificar más fácilmente la causa del error, pero los errores lógicos

son más difíciles de localizar y corregir porque la única información disponible sobre el error, es la discrepancia entre el output real y el esperado. Así pues para los errores lógicos, es importante que los programadores entiendan la función total global del programa, pero también tienen que tener un entendimiento muy detallado de lo que el programa hace sentencia a sentencia. Por tanto para los errores lógicos en la depuración, parece probable que el nivel de comprensión del programa separa claramente a los programadores expertos de los novatos.

Este estudio también indicó que las razones principales de la superioridad de los programadores expertos en localizar y corregir errores lógicos eran su:

a.- Habilidad para aislar el error a un segmento del programa.

b.- Habilidad de seleccionar la hipótesis correcta.

Los programamdores novatos, por otra parte, tenían dificultad en aislar el error, al segmento del programa debido. Por ello, corrigieron sólo, unos pocos errores lógicos y emplearon más tiempo en depurar el programa que los expertos.

Incluso aunque encontramos diferencias significativas en la comprensión entre programadores expertos y novatos, este estudio no revela el alcance completo de la habilidad de depuración de los programadores expertos, porque ellos depuraron un programa relativamente pequeño, de un dominio de problemas, conocido para ellos.

CAPITULO 5

UN MODELO DE COMPORTAMIENTO EN LA DEPURACION PARA PROGRAMADORES EXPERTOS Y NOVATOS

5.1. Introducción

Los investigadores de factores humanos en programación, han utilizado nociones cogidas de la psicología cognoscitiva y de la resolución de problemas por medios informáticos, para producir modelos conceptuales de comportamiento de un programador, para varias tareas de programación.

Sin embargo, sólo unos pocos investigadores propusieron modelos cognoscitivos, para el comportamiento en la depuración de programas. Por ejemplo [Goul75], [Kess86], describieron modelos de cómo se detectan errores en los programas. Shneiderman, desarrolló un modelo genérico para explicar el comportamiento, para tareas

básicas de programación tales, como la composición, la comprensión, la depuración, la modificación y el aprendizaje del programa [Shne77].

Este capítulo presenta un modelo conceptual, del comportamiento en la depuración, para programadores estudiantes novatos y expertos que se basa en los resultados de los 3 experimentos descritos en los capítulos 3 y 4. El modelo explica el comportamiento clásico en la depuración, de programadores en PASCAL expertos y novatos. El modelo asume, que el programa contiene un número desconocido de errores semánticos y lógicos.

El modelo de depuración propuesto en este capítulo, difiere de los modelos anteriores en 3 puntos principalmente:

a.- Explica cómo los errores de los programas, se detectan, de la misma forma que se corrigen.

b.- Describe el comportamiento en la depuración, tanto de los programadores expertos como de los novatos.

c.- Ve la tarea de depurar un programa, como si de una actividad de comprensión se tratara.

5.2. Modelos de depuración anteriores

Esta sección resume brevemente los modelos de comportamiento en la depuración, de los que se habla en la literatura.

El modelo de Gould, está basado en su investigación sobre cómo programadores experimentados, hallaron los errores no sintácticos insertados en un programa FORTRAN. En este modelo, Gould decía que en la estrategia de los expertos, estos primero buscaban indicaciones sobre los errores, si encontraban una pista que podía llevarles a la detección de un error, generaban una hipótesis sobre por qué se producía el error y luego evaluaban la hipótesis. Bajo ciertas circustancias tales como un fallo repetitivo de una estrategia de depuración concreta, el experto echaba mano de una estrategia diferente. Gould también decía que los expertos buscaban primero los errores sintácticos y luego los errores no sintácticos [Goul75]. Sin embargo, no queda claro hasta qué extremo el modelo de Gould describía la forma en que los novatos detectaban los errores.

Kessler, desarrolló un modelo de depuración basado en los datos de protocolo y propuso una simulación de la depuración que efectúa el novato, utilizando reglas de producción. Su modelo computacional especificaba

completamente, los procesos necesarios para depurar funciones LISP de una línea. Esto lo realizaba descomponiendo los procesos de depuración en cuatro fases: Comprension, detección del error, localización del error y corrección del error [Kess86]. Puesto que este modelo se ponía en práctica, como un sistema de producción con encadenamiento hacia adelante, no describia los procesos reales, llevados a cabo por los novatos en cada fase, pero explicaba algunas de las dificultades con las que los novatos se encontraban cuando depuraban funciones LISP.

Shneiderman, desarrolló un modelo cognoscitivo del comportamiento de un programador, basado en un número de experimentos psicológicos controlados. Schneiderman dijo que los programadores experimentados, desarrollaron unos conocimientos complejos de varios niveles, sobre los conceptos y técnicas de programación durante la depuración [Shne77].

El modelo separaba el conocimiento sintáctico del semántico. El conocimiento sintáctico consiste en una información más precisa y detallada sobre el lenguaje de programación, tal como el formato de iteración, el bloque condicional o sentencias de asignación, conjuntos de caracteres válidos y los nombres de funciones de biblioteca. El conocimiento semántico, consiste en conceptos generales de programación que son independientes del lenguaje de programación utilizado. Va desde el nivel

bajo de comprensión de varias sentencias de programa, a estrategias de mayor nivel para producir un programa específico. Este modelo también incluye una memoria de largo plazo del conocimiento sintáctico y semántico y una memoria de trabajo para la construcción de soluciones de problemas.

Finalmente, Schneiderman sugirió que la representación interna de un programa, creada por los programadores era muy crítica, para el éxito de la depuración.

5.3. Apoyo empírico del modelo

Nuestra motivación original de proseguir los experimentos de depuración descritos en los capítulos 3 y 4, era ver cómo la comprensión del programa por parte de los sujetos afecta a la depuración, identificar qué tipo de procesos llevan a cabo los sujetos durante la depuración, y señalar exáctamente por qué los programadores expertos se diferencian de los programadores novatos.

Los resultados obtenidos basados en estos experimentos, nos animaron a desarrollar un modelo de depuración. La base del modelo de depuración descrito en este capitulo, viene en parte de estudios anteriores de depuración de programas, pero mayoritariamente surge de los resultados de los 3 experimentos de depuración descritos en los capítulos 3 y 4.

5.4. *Un modelo del comportamiento en la depuración*

Los modelos descriptivos generales del comportamiento en la depuración de programadores expertos y novatos se ilustran en las figuras 5.1.a y 5.1.b respectivamente. Asumen, que la clave para la detección y corrección del error, es la comprensión del programa por parte de los programadores. Por comprensión, queremos decir, construir una descripción mental de lo que el programa hace y cómo lo hace. Las secciones siguientes describen el modelo descriptivo con detalle y muestran las diferencias entre expertos y novatos.

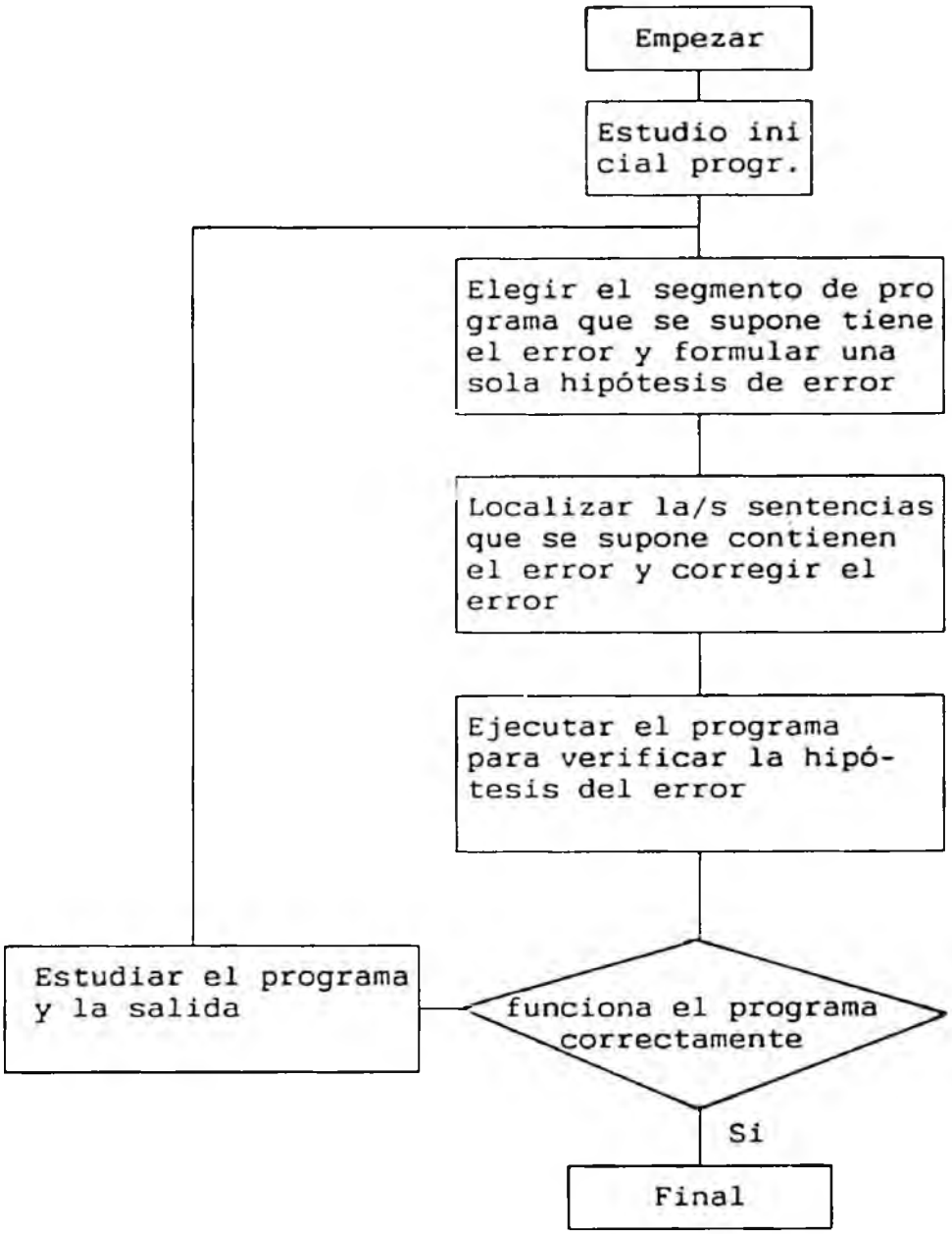
Empezar
Estudio ini cial progr.
Elegir el segmento de pro grama que se supone tiene el error y formular una sola hipótesis de error
Localizar la/s sentencias que se supone contienen el error y corregir el error
Ejecutar el programa para verificar la hipó- tesis del error
funciona el programa correctamente
Estudiar el programa y la salida
Si
Final

Figura 5.1a

Un modelo descriptivo del comportamiento de los expertos en la depuración

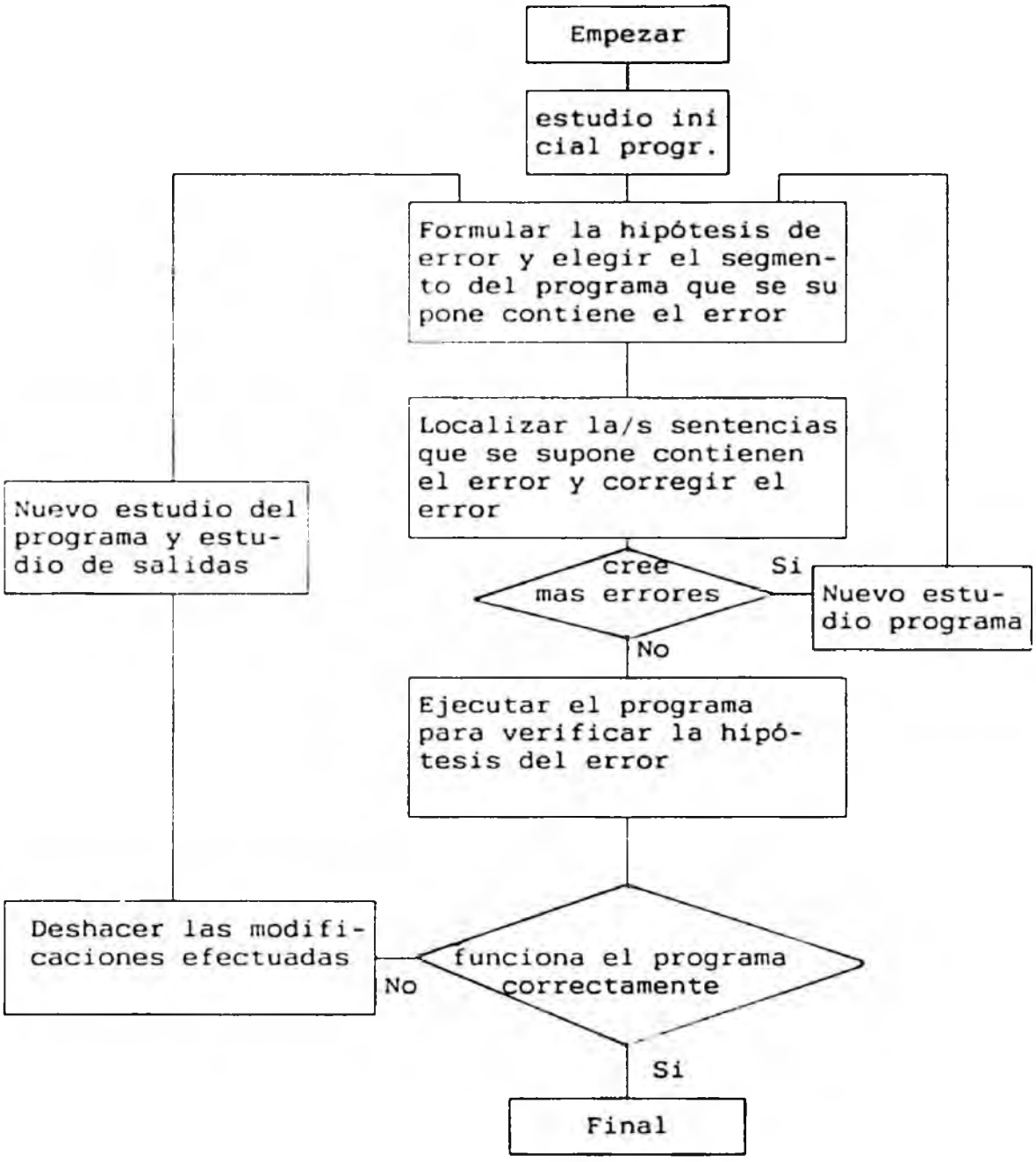

Figura 5.1b

5.4.1. Estudio inicial del programa

Los programadores expertos y novatos empezaban su sesión de depuración, estudiando el programa defectuoso o bien en la pantalla o bien en el listado.

Generalmente leían el programa para hacerse una idea sobre lo que se supone que el programa hacía. Unos pocos novatos del experimento 1, ejecutaron el programa rápidamente casi sin leerlo, probablemente porque estaban buscando un mensaje de error explícito del sistema. Una gran diferencia entre programadores expertos y novatos durante el estudio inicial era el orden en el que leían el programa. Los expertos leían el programa en el orden en que realmente iba a ser ejecutado - primero el programa principal y luego los procedimientos llamados por el programa principal, en el orden auténtico de proceso.
Los novatos leían el programa de principio a fin secuencialmente. Esto nos hacía pensar que los expertos primero intentaban obtener una comprensión a un nivel mas alto de como funcionaba el programa, mientras que los novatos usaban una lectura secuencial para entender el programa. Además, los novatos a menudo simulaban manualmente, la ejecución de uno o más de los procedimientos del programa, tomando notas sobre los valores de las variables en los diferentes estados -partes- del programa, mientras que los expertos a menudo empleaban la ejecución mental de los segmentos del programa.

Además del listado del programa, los programadores a menudo estudiaban el siguiente material suplementario:

a.- Listado del archivo de datos de entrada.

b.- Listado de la salida real (incorrecta).

c.- Listado de la salida esperada (correcta)

d.- Mensajes de error del sistema.

Los expertos y los novatos usaban las mismas fuentes de información durante el estudio inicial del programa, pero como se muestra en los experimentos 2 y 3, los expertos tenian más éxito a la hora de obtener una comprensión global del programa y a la hora de generar una hipótesis inicial de alta calidad.

Aunque [Guge86], nos informó de una observación similar, nuestros resultados experimentales parecían indicar, que los expertos generalmente empleaban más tiempo en la lectura inicial del programa que los novatos. La cantidad de tiempo que los programadores empleaban en su lectura inicial estaba hasta cierto punto, influenciada por el tipo de error del programa.

Por ejemplo, un mensaje de error semántico lleva a menudo, a que los novatos pasen bastante rápido al próximo

paso, esto es, a aislar el error, al número de sentencia indicado por el mensaje de error.

Como resultado de lo anterior se podía deducir que ellos empleaban menos tiempo, en la lectura inicial del programa. Por otra parte, un error lógico les animaba o bien a evaluar la discrepancia entre las salidas real y esperada, o bien a leer el programa para obtener una comprensión total del programa. Puesto que el objetivo de muchos expertos, durante la lectura inicial, era entender la función total del programa de manera que ellos pudiesen obtener las pistas de los errores, parecía que ellos tardaban más tiempo en la lectura inicial, que los novatos.

Además del tema de la comprensión, los programadores tenían en cuenta, el obtener pistas sobre los errores, a partir del estudio inicial de los materiales habidos. Comparaban el output real con el esperado en búsqueda de posibles discrepancias y estudiaban cuidadosamente el mensaje de error del sistema. Sin embargo, si el output real del programa defectuoso no estaba disponible o si sólo se conocía el mensaje de error, ellos procesaban siempre el programa, para observar el estado real del programa. Al examinar el fichero de entrada, los programadores reunían información sobre el número y el tipo, de datos de entrada y la organización de estos datos. Además, los mensajes de error del sistema a menudo suministraban pistas valiosas para encontrar los errores.

Por ejemplo, un mensaje de error tal como "el valor de una variable o de sub-expresión está fuera de rango de su uso deseado" a menudo incitaba a los programadores a ejecutar una o ambas de las acciones siguientes:

a.- Verificar si la matriz dada se declaraba con las dimensiones adecuadas.

b.- Verificar si el valor del índice se calculaba o manipulaba correctamente.

Los programadores reunían información de las características del "estilo" del programa tales como el propósito general del programa (comentarios globales), descripción de los segmentos o sentencias del programa individual (comentarios entre-líneas), nombres de variables nemotécnicos (significativas), indentación del programa, y modularización del programa.

La evidencia de esta observación venía de los datos de protocolo de los sujetos, obtenidos en nuestros experimentos 2 y 3. Por ejemplo, tres novatos y un experto en el experimento 2 y tres novatos y dos expertos en el experimento 3, dibujaron corchetes en el lado izquierdo del listado del programa, para marcar la estructura del bloque. Los novatos y los expertos también se referían frecuentemente a los comentarios globales y a la parte de declaración del programa.

Durante el estudio inicial, los novatos a menudo concentraban su atención en un sólo segmento del programa, mientras que los expertos no descendían tanto, sino que se organizaban a nivel de bloque en trozos útiles. Esta observación era bastante evidente según la tarea de reconstrucción, encargada a los sujetos y descrita en los experimentos 2 y 3. La mayor habilidad de troceado de los expertos explicaba parcialmente, por qué tenían más éxito y eran más eficientes en localizar y corregir errores. [Vess85], [Wied86], también explicaban, que los expertos se construían una representación abstracta del programa, durante la depuración y comprensión del mismo.

Con estas observaciones, era bastante razonable llegar a la conclusión, de que los expertos obtenían una comprensión total de las funciones del programa, junto con algún conocimiento a nivel de sentencia, mientras que los novatos obtenían una comprensión a nivel de sentencia de las diferentes partes del programa, pero no una buena comprensión total del mismo.

5.4.2. Hipótesis de error y selección del segmento posible

El objetivo del proceso de generación de hipótesis, es utilizar las pistas disponibles o heurísticas para aislar el error a un cierto segmento del programa y después aislar el error a una sentencia o sentencias específicas. El proceso de generar una hipótesis de error de buena calidad, sin embargo, planteaba un problema mayor para los programadores novatos, especialmente en los errores lógicos. El resto de esta sección, explica en detalle cómo los programadores generan hipótesis de error y cómo eligen el segmento del programa que se cree que contiene el error.

Como está ilustrado en la figura 5.1.a y 5.1.b. los novatos y los expertos difieren en el orden en el que ejecutan los siguientes pasos:

a.- Generar la hipótesis.

b.- Elegir el segmento que se cree que contiene el error.

Los novatos hacen b y luego a, mientras que los expertos hacen a y luego b. La pobre comprensión que tenían los novatos del programa, mas los mensajes de error del sistema a menudo les animaba a seleccionar el segmento del programa primero y luego generaban una hipótesis de error.

Los novatos podían seleccionar mas fácilmente un segmento del programa, que se creía que es el que causa el error semántico, porque cuando se ejecuta el programa el compilador del lenguaje usualmente visualiza el mensaje de error, junto con una indicación de la sentencia donde ocurre el error. Esto era la estrategia general de selección del segmento del programa, que solían usar tanto los novatos como los expertos.

La tarea de selección del segmento era más difícil para un error lógico, que para uno semántico, porque el sistema no suministra información explícita sobre los errores lógicos, por tanto los novatos tenían que determinar, qué procedimiento o función contiene el error lógico, basado en la discrepancia existente que hay entre las salidas reales y las esperadas.

Nuestras observaciones de los novatos durante los experimentos de protocolo, indicaron que tendían a usar las dos siguientes estrategias generales de aislamiento para los errores lógicos:

a.- Estrategia de prueba - y - error.

b.- Estrategia heurística.

La estrategia de prueba - y - error consiste en seleccionar un procedimiento o función al azar. Es una

estrategia que consume mucho tiempo y es ineficaz. La pobre comprensión del programa y/o la incapacidad de encontrar una pista sobre el error, llevaba a los programadores novatos a adoptar esta estrategia. Los novatos recurrían a esta estrategia, cuando todas las demás estrategias fallaban.

La estrategia heuristica, consiste en seleccionar un procedimiento o función basado en reglas de sentido común, tal que se selecciona la rutina más compleja o el segmento del programa mas grande. Aunque habia otros dos errores en el programa, casi todos los novatos de los experimentos 2 y 3 primero seleccionaron el segmeto que efectuaba la búsqueda-binaria. Es importante anotar que los novatos continuaban concentrándose en el segmento que buscaba los valores, incluso después de hacer varias modificaciones y ningún progreso.

Después de que se seleccionaba un segmento del programa, los novatos formulaban una hipótesis de error. Se supone que la hipótesis de error inicial se creaba principalmente, después del estudio inicial del programa. Las hipótesis de error posteriores se generaban normalmente, tanto leyendo el programa, como examinando la salida de las ejecuciones previas. Aunque el nivel de detalle de la hipótesis de error variaba entre los programadores expertos y los novatos, al menos relacionaba los datos de entrada, el oupput real, las principales

estructuras de los datos del programa, los datos globales y los algoritmos usados en el programa.

Nuestros estudios de protocolo sugirieron, que los novatos empleaban frecuentemente las siguientes estrategias generales de depuración, para formular una hipótesis de error:

a.- Prueba - y - error.

b.- Simulación manual del segmento del programa.

c.- Incluir sentencias write donde lo crean oportuno.

d.- Backtracking (marcha atrás).

Gugerty analizó algunas de las estrategias de los novatos en [Guge86]. La estrategia más común de los novatos era incluir sentencias Write.

En la estrategia de prueba - y - error los novatos no tenían una idea clara sobre los errores y rápidamente se sumergían en la experimentación con el programa, ya que no tenían un entendimiento profundo del programa.

En la estrategia de simulación manual, los novatos simulaban la ejecución de un segmento aislado del programa, tomando notas sobre los valores de la variable en varios

lugares del segmento del programa, rastreando el flujo de control del segmento y dibujando los diagramas de estructuras de datos. También hacían referencia a las variables globales de la parte de declaración del programa, a los parámetros del procedimiento, a la llamada del segmento del programa seleccionado y a todos los procedimientos que eran llamados, por el segmento del programa seleccionado.

En la estrategia de incluir sentencias Write, los novatos insertaban estas, en varios lugares del segmento del programa para seguir la pista de una o más de las variables principales. Los programadores novatos insertaron muchas más que los programadores expertos. Además generaron una salida considerable, porque insertaron sentencias Write en lugares inapropiados como por ejemplo dentro de un bucle y rastreaban los valores de las variables que no estaban implicadas directamente en el error.

En la estrategia de backtracking, los programadores trabajaban hacia atrás, desde los resultados incorrectos (síntoma), a través de la lógica del segmento del programa y hasta que descubrían el lugar, donde la lógica del programa era incorrecta. Esta estrategia implicaba simulación manual inversa del programa, desde donde el error ocurría o donde la discrepancia entre la salida real y la salida esperada aparecía. En nuestros experimentos,

backtracking era la estrategia más comunmente usada, por los novatos para localizar un error semántico.

En contraste con los aprendices, los programadores expertos generaban hipótesis de error, antes de que seleccionaran el segmento del programa que se creía que contenia el error. Esto es así porque probablemente su acercamiento a la comprensión del programa para depurar, les permitía formar hipótesis sobre el error. Además de algunas de las estrategias, sentencias write y backtracking usadas por los aprendices, los expertos usaban las dos estrategias siguientes para generar la hipótesis del error:

a.- Ejecución mental del segmento del programa.

b.- Ayudas de depuración ON-LINE.

Al igual que los novatos, los expertos también usaban sentencias write en el programa. Sin embargo, insertaron muchas menos, generalmente al final del procedimiento o función. Además, los expertos empleaban una estrategia de backtracking, para generar hipótesis de error, no sólo cuando obtenían el mensaje de error del sistema, sino también cuando terminaba el programa anormalmente. El proceso de simulación del programa de los expertos era puramente mental.

Otra estrategia importante de generación de hipótesis de error de los expertos,era el uso de las herramientas de depuración ON-LINE. Muchos sistemas de ordenador suministran ayudas de este tipo, para obtener la información en tiempo real de la ejecución del programa. Estas herramientas proporcionan varias características tales como la ejecución paso por paso del programa, el rastreo de las variables y la ejecución parcial del programa.

Los programadores expertos a menudo utilizaban herramientas ON-LINE, para obtener más imformación sobre la ejecución del programa. Pero es importante advertir que muchos programadores expertos, utilizaron estas herramientas mas para una mera reunión de información que como una herramienta de solucionar problemas de forma directa. No sabemos por qué los novatos eran reacios a utilizar las herramientas ON-LINE. Las razones más probables son:

a.- Falta de conocimiento sobre estas ayudas.

b.- Falta de habilidad para usar estas herramientas.

c.- Consideraron que estas herramientas no eran "amistosas".

d.- Eran incapaces de usar estas herramientas eficazmente.

Los programadores expertos se diferencian de los programadores novatos, en que los primeros usaban varias estrategias de depuración. De nuestro experimento 1 en los datos de protocolo vemos cómo usaban una combinación de estas estrategias durante la depuración. Por ejemplo, casi todos los expertos del experimento 1 usaron ayudas de depuración ON-LINE, junto con la ejecución mental del segmento del programa. Una observación más que se puede hacer en los 3 experimentos, es la calidad superior de las hipótesis de error generadas. Como resultado de todo esto se puede decir, que corrigieron más errores haciendo sólo unas pocas modificaciones.

Después de haber formado una hipótesis inicial sobre el error, los expertos elegían el procedimiento o función que creían que contenía el error. La selección de un procedimiento apropiado o función era vital para una depuración eficaz del programa. La selección correcta de un procedimiento, a menudo depende de la comprensión total del programa, que obtienen los programadores durante el estudio inicial del mismo, y el tipo de error (semántico o lógico) que encuentran a la hora de ejecutar el programa.

Con el propósito de elegir el segmento del programa que se creía que contenía el error, los expertos

seleccionaron o bien la estrategia heurística o la estrategia sistemática. Nuestros datos de protocolo revelaron, que la estrategia heurística de los expertos era similar a la usada por los novatos. Generalmente, el experto adopta esta estrategia, pero cuando falla tiende a cambiar a otras estrategias.

La estrategia sistemática consiste, en depurar el programa en una forma de orden real de ejecución - el programa principal primero, luego el primer procedimiento llamado por el programa principal, después el segundo procedimiento llamado y así sucesivamente. En esta estrategia los expertos insertan sentencias Write, para seguir la pista de los valores intermedios de ciertas variables clave en lugares cuidadosamente seleccionados del subprograma. Las partes del programa que no han sido depuradas aún, estan ausentes o se modifican como versiones falsas (por ejemplo, líneas de comentarios). Los programadores también usan herramientas de depuración ON-LINE, para observar el estado del programa o para rastrear valores de variables. Esta es una estrategia muy efectiva que sólo usaban los expertos.

5.4.3. Localizar y corregir errores

Una vez que el error había sido localizado, los programadores intentaban corregirlo modificando una o más sentencias del programa o de los segmentos del programa. Los programadores a menudo modificaban más de una sentencia del programa, para corregir cada error en el programa o reescribían totalmente el segmento del programa, porque sabían cómo codificar estos algoritmos de una manera muy concreta. En contraste con los novatos, los programadores expertos reparaban cada error cambiando muy pocas sentencias del programa.

Los programadores inicialmente parecían suponer, que el programa defectuoso contenía un único error. Mientras buscaban pistas sobre el error, sin embargo, la comprensión total del programa por parte de los expertos, a menudo les ayudaba a localizar otros errores. Una vez que los expertos se daban cuenta de que el programa contenía varios errores, leían el programa y miraban el resto de materiales con más cuidado, concentrando su atención en los detalles mas concretos o volviendo a evaluar su estrategia de depuración. En el experimento 1, como los errores semánticos era relativamente más fáciles de localizar que los errores lógicos, encontraron frecuentemente estos errores primero. Como resultado de lo anteriormente mencionado se puede deducir que primero tendían a corregir todos los errores juntos y luego verificaban estas

correcciones. Esta estrategia de corrección de errores múltiples de los expertos demostraba su confianza en hacer las correcciones debidas.

De los datos de protocolo verbal del experimento 1, se ve claramente que en los errores semánticos, los programadores aprendices y los expertos parece que comprendían el mensaje de error igualmente bien, pero sin embargo los programadores expertos tenían más probabilidades de generar hipótesis correctas. Esto podía ser debido a la experiencia previa con la misma clase de errores, y/o a su comprensión mas global del programa. La mayoria de las veces, la hipótesis de error de los programadores novatos era inferior a la de los programadores expertos, especialmente para los errores lógicos. Se supone que la pobre comprensión que tenían del programa explicaba parcialmente este comportamiento.

Cuando buscaban un error con varias causas posibles, los programadores novatos se centraban sólo en una hipótesis cada vez, hasta que se verificaba o se rechazaba la hipótesis. Desde el momento que los programadores expertos usaban el método de comprensión para generar hipótesis para tratar de corregir los errores semánticos y los lógicos, a menudo tenían varias hipótesis para un único error o varias hipótesis para varios errores y ellos cambiaban las hipótesis mientras descubrían pistas.

Nuestros resultados de los experimentos 2 y 3 indicaron que si el programa contenía sólo errores lógicos, los expertos no podían usar la estrategia de corrección de múltiples errores. Quizá, esto puede ser debido al hecho de que los errores lógicos son normalmente más difíciles de corregir que los errores semánticos. Otra posibilidad sin embargo, era que los experimentos 2 y 3 contenían sólo errores lógicos.

5.4.4. Ejecución del programa para verificar hipótesis de error

El propósito de este paso en el proceso de depuración era determinar si las modificaciones corregían los errores. Sin embargo los novatos y expertos diferían en cuando ejecutar el programa. Los novatos siempre realizaban esta tarea, inmediatamente después de hacer las primeras modificaciones para corregir un único error, mientras que los programadores expertos ejecutaban el programa, después de haber hecho las suficientes modificaciones como para corregir varios errores a la vez. El comportamiento de los sujetos en el Experimento 1 demuestra que los expertos trabajaban con varias hipótesis al mismo tiempo, mientras que los novatos sólo trabajaban con una única hipótesis cada vez.

Cuando las sentencias modificadas en el programa introducían errores adicionales, los expertos los corregían inmediatamente, deshaciéndo las modificaciones efectuadas previamente. Sin embargo los novatos cuando eso sucedía, no siempre eran capaces de dejar el programa, tal como estaba antes de las modificaciones, por lo cual ellos frecuentemente incorporaban nuevos errores al programa.

Gugerty, comentaba que sus sujetos introducían errores adicionales durante la depuración, pero no era capaz de explicar por qué ocurría eso [Guge86]. Nuestro modelo explica en parte por qué los novatos introducían muchos errores en el programa.

Después de cada ejecución del programa, los programadores tomaban alguna de las siguientes decisiones, basadas en los mensajes de error del sistema o en las salidas obtenidas:

a.- Seleccionar un nuevo segmento de programa.

b.- Seleccionar una nueva estrategia.

c.- Seleccionar una nueva hipótesis de error.

Nuestras observaciones sobre los sujetos indican que los expertos decidían una estrategia diferente, cuando se veían incapaces de localizar y corregir un error.

Por otro lado en cambio, los novatos por lo general utilizaban la estrategia de aislamiento, independientemente de los mensajes de error y de las salidas que obtuviesen. Lo habitual era que los novatos cada vez que obtuviesen un mensaje de error después de cada ejecución del programa, siguiesen ciñéndose al segmento de programa donde se producía el mensaje de error. Esto indica que los novatos tenían una dependencia total, de los mensajes de error que el sistema iba produciendo.

5.4.5. Estudio del programa y de sus salidas

El objetivo que debían perseguir los programadores, era el de estudiar todos los listados que se obtuviesen, con el fin de incorporar nueva información, para lograr una más completa comprensión del funcionamiento del programa, pues ellos estudiaban a veces defectuosamente, el programa y otros materiales importantes, durante la sesión de depuración alargando la misma pues tenían que conseguir de todas formas corregir los errores. A menudo ellos consideraban clave, información como la naturaleza del error y la organización de los datos de entrada, pero el estudio del programa era necesario. Los programadores raramente usaban datos de entrada durante los estudios posteriores del programa. Estos estudios y el estudio previo se parecían bastante, pero hay una diferencia importante y es que en el estudio posterior del programa,

los programadores bajaban en la comprensión a niveles de detalle más concretos, pero también aquí es distinto para expertos y para novatos, pues los primeros se fijaban en varios segmentos de programa y los segundos sin embargo concretan su atención en un sólo segmento.

5.5. *Diferencias entre expertos y aprendices*

Nuestro estudio sobre la depuración de programas y otros muchos anteriores, [Guge86] entre otros, identificaron las principales diferencias entre programadores expertos y aprendices. Algunas de estas diferencias mas otras encontradas en los experimentos 1, 2 y 3 se resumen en la tabla 5.1. El resto de esta sección explica brevemente algunas de estas diferencias con respecto al modelo de depuración propuesto aquí.

Resúmen de las diferencias expertos/aprendices

Características	Aprendices	Expertos
Comprensión inicial	Orden físico de ejecución	Orden real de ejecución
Estrategia depuraci	Aislamiento	Comprensión
Corrección de error	Error único	Errores múltiples
Orden de corrección del error	1º errores semánticos luego errores lógicos	Errores semánticos y lógicos a la vez
Calidad de la 1ª hipótesis	Inferior	Superior
Tipo de error	Corrigen sólo los errores semánticos	Corrigen los errores semánticos y lógicos
Estrategias de aislamiento y error	Prueba y error Mensajes de error del sistema	Sistemática Mensajes de error del sistema y Heurística
Estrategias de depuración	Prueba y error Simulación manual Sentencias Write Backtrack	Ayudas de depuración on-line Ejecución mental Write y backtrack
Habilidad de troceado del programa	Inferior	Superior
Utilización de ayudas de depuración on-line	No	Si
Reescritura del programa	Si	No
Método comprensión	Simulación manual	Ejecución mental
Quitar modif. malas	No	Si
Tiempo depuración	Alto	Bajo
Nº de ejecuciones	Alto	Bajo
Nº modificaciones	Alta	Baja
Nº error corregidos	Corrigen pocos	Corrigen todos
Nº de errores nuevos añadidos	A menudo introducen errores al depurar	Rara vez generan errores al depurar

Tabla 5.1

Descubrimientos anteriores en la depuración de programas nos muestran, que los programadores expertos localizaban más rápido los errores, que los programadores novatos, pero no especulaban sobre por qué sucede así. Al igual que en estos estudios, los resultados de nuestros experimentos 1, 2 y 3, también indicaron que los programadores expertos eran más eficientes, que los programadores aprendices en localizar y corregir errores. Además, parece que nuestro modelo puede explicar esta diferencia, que puede ser debida a las siguientes observaciones:

a.- Los expertos hacían muchas menos modificaciones al programa que los novatos.

b.- Los expertos ejecutaron el programa con menos frecuencia que los novatos.

c.- Los expertos introdujeron menos errores que los novatos al hacer modificaciones.

d.- Los expertos localizaron y corrigieron errores en conjunto, los novatos lo hacían de uno en uno.

Otra gran diferencia entre programadores expertos y aprendices era, la manera en que localizaban, corregían y verificaban los errores. Todos los expertos del experimento 1 localizaban y corregían varios errores,

mientras que todos los novatos localizaban y corregían sólo un error cada vez. Esta observación de los sujetos muestra, que los expertos trabajaban con hipótesis de múltiples, por ello, los expertos necesitaban tener una comprensión más profunda del programa.

Parece como si hubiera dos acercamientos generales de depuración, que los sujetos intentaban emplear durante la sesión de depuración:

a.- Acercamiento de comprensión.

b.- Acercamiento de aislamiento.

En el acercamiento de comprensión, los programadores intentaban localizar el error, mediante la primera comprensión de lo que el programa hacía realmente, comparado con lo que se supone que debía hacer. La comprensión total del programa ayudaba a los programadores a identificar fácilmente las causas de los errores.

En el acercamiento de aislamiento, los programadores intentaban identificar inmediatamente, la localización del error, buscando pistas en la salida, incurriendo en errores similares, probando estados del programa o usando el conocimiento del dominio de la aplicación.

La selección de este acercamiento llevaba a los programadores frecuentemente a sumergirse directamente en el programa. Los programadores que empleaban el acercamiento de comprensión, pasaron más tiempo en el estudio inicial del programa, que aquellos que usaron el acercamiento de aislamiento. Los programadores que emplearon el acercamiento de comprensión, eran más eficientes y tenian más éxito en localizar y corregir errores (sobre todo los de tipo lógico), que los programadores que usaron el acercamiento de aislamiento. Nuestros datos de protocolo de los experimentos 1, 2 y 3, indicaron que los expertos empleaban el acercamiento de comprensión en la depuración, mientras que los novatos usaban un acercamiento de aislamiento.

Los programadores expertos y novatos estudiaban los materiales de información disponibles, formaban hipótesis de error, elegían el segmento del programa, que creían que contenía el error, modificaban una o más sentencias del programa, y procesaban el programa para ver el efecto de los cambios. Los programadores expertos, eran más eficaces en ejecutar estos pasos. Ellos sabían también cuando usar uno de estos pasos. Por ejemplo, cuando las sentencias modificadas introducían errores adicionales, todos los expertos de nuestros experimentos restauraron el programa inmediatamente al estado anterior. Esto les permitía quitar el error que ellos introducían, inmediatamente y formar una hipótesis nueva o refinar la hipótesis previa.

Los programadores expertos se diferenciaban también de los aprendices, en seleccionar el segmento del programa que se creía que contenía el error. Después del estudio inicial del programa, los expertos primero formaban una hipótesis sobre el error y luego seleccionaban un segmento del programa, mientras que los programadores novatos seleccionaban un segmento del programa, basados en un mensaje de error explícito y luego generaban una hipótesis sobre el error. El orden en el que ellos ejecutaban estos dos pasos, no podía ser tan crítico para los errores semánticos, pero sin embargo si lo era para los errores lógicos, con lo que éxito en la depuración estaba muy influenciado por el mencionado orden de ejecución.

5.6. Resumen

En resumen, el modelo propuesto entiende el comportamiento de los programadores expertos y novatos en la depuración, fundamentalmente como un proceso de comprensión de un programa. Describe en detalle los procesos comunes que están implicados en la depuración de programas. También identifica las siguientes diferencias principales entre programadores expertos y novatos:

a.- Los programadores expertos parece que emplean el acercamiento de comprensión, mientras que los programadores novatos, intentan usar el acercamiento de comprensión, pero luego cambian al acercamiento de aislamiento, concentrándose en un procedimiento o función si el programa está modularizado o en un determinado segmento del programa monolítico si no lo está.

b.- Los expertos quitan inmediatamente los errores que introducen involuntariamente, deshaciendo las modificaciones previas, mientras que los novatos no restauran el programa al estado previo inmediatamente porque no son capaces de deahacer las modificaciones efectuadas.

c.- La hipótesis de error de calidad superior de los programadores expertos les ayuda a reparar el error, modificando unas pocas sentencias y por tanto con menor peligro de introducir errores adicionales, mientras que la hipótesis de error, de inferior calidad de los programadores novatos, les induce a modificar muchas sentencias e introducen muchos errores adicionales que les cuesta mucho corregir.

CAPITULO 6

CONCLUSIONES Y FUTUROS TRABAJOS

6.1. Conclusiones

Esta investigación descubre algunas interioridades, sobre el proceso de la depuración de programadores expertos, de nivel medio y novatos.

Los principales resultados se resumen como sigue:

a.- Los expertos utilizan una estrategia de comprensión, gracias a la cual primero intentan comprender el programa y luego usar ese conocimiento adquirido, para encontrar los errores mas fácilmente. Los de nivel intermedio y los novatos emplean una estrategia de aislamiento, por la que intentan identificar inmediatamente las localizaciones de los errores, buscando en las salidas las pistas necesarias para ello, recordando errores similares vistos anteriormente y probando los diferentes estados del programa.

b.- Los expertos corrigen varios errores a la vez, antes de verificar las correcciones, mientras que los de nivel medio y los novatos corrigen y verifican los errores de uno en uno. Los de nivel medio y los novatos corrigen primero los errores semánticos y luego los errores lógicos, mientras que los expertos corrigen ambos, los errores semánticos y los lógicos, a la vez.

c.- Los expertos son más rápidos y tienen más éxito en corregir todos los tipos de errores, modifican menos sentencias e introducen menos errores nuevos, que reparan inmediatamente. Los novatos no corrigen todos los errores, hacen modificaciones muy extensas e introducen muchos errores nuevos. Los de nivel medio corrigen todos los errores, pero haciendo un número considerable de modificaciones y también introducen varios errores nuevos.

d.- Los expertos emplean más tiempo en la lectura inicial del programa, que los novatos y por lo tanto tienen una mayor comprensión sobre lo que el programa tiene que hacer, que los novatos. Por ello los expertos son capaces, de generar una hipótesis inicial de mayor calidad, sobre la posible causa del error.

e.- La comprensión del programa puede no ser tan útil para localizar y corregir errores semánticos, pero es imprescindible para localizar y corregir los errores lógicos, siendo importante que los programadores entiendan, no sólo todo el funcionamiento global del programa, sino también la información a nivel de sentencia.

f.- La superioridad de los expertos en localizar y corregir errores, se debe en definitiva a su mejor:

1.- Comprensión total del programa.

2.- Habilidad para aislar el error, al segmento del programa, causante del mismo.

3.- Habilidad para fabricar una hipótesis sobre el error, correcta.

6.2. *Limitaciones del estudio*

Hay algunas limitaciones para la generalización de los resultados explicados aqui.

Primero, los experimentos eran una prueba de la depuración y comprensión del programa, en una situación experimentalmente controlada.

Segundo, los sujetos depuraron programas escritos por el investigador, en vez de sus propios programas. Mientras este es un hecho bastante común, para los programadores profesionales, no sucede asi para la mayoría de los programadores estudiantes, que estan acostumbrados a depurar únicamente sus propios programas

Tercero, investigamos la depuración de programas cortos orientados a los estudiantes, en lugar de programas extensos "del mundo real". Se requerirán otros experimentos, para determinar si se pueden obtener resultados similares, en la depuración de programas más amplios y no orientados al mundo universitario.

Cuarto, clasificamos la experiencia en la depuración de los sujetos, basándonos en los cursos de programación a los que habían asistido. Habría que confirmar si este criterio de clasificación, es válido para otro tipo de

programadores, que no pertenezcan al mundo universitario.

6.3. Futuros trabajos

El trabajo explicado en esta tésis representa sólo, un pequeño paso en el camino de la comprensión de la depuración de programas. Se suscitan muchas preguntas de cara a posibles futuras investigaciones sobre la depuración de programas. Se citan a continuación, unas cuantas de esas preguntas:

- Estrategias de depuración

¿ Se usan tipos de estrategias de depuración diferentes de las de comprensión y aislamiento ?. Y, si es que sí. ¿ Cuales son ?.

¿ Usan los programadores profesionales ese mismo tipo de estrategias ?.

¿ Hasta qué punto son comunes estas estrategias ?.

¿ Hasta qué punto es dependiente la estrategia utilizada, de la información disponible ?. Y, ¿ de la persona ?. Y, ¿ del dominio del problema del que se trate ?.

- **Clasificación de la experiencia en la depuración**

¿ Qué criterios de clasificación serían buenos para catalogar la experiencia de los programadores, diferentes de los del número de años de experiencia y número de cursos realizados ?.

- **Errores**

¿ Son diferentes las estrategias utilizadas si un programador sabe si hay uno o varios errores, de antemano ?.

- **Factores del programa**

¿ Qué importancia tienen los factores del "estilo" del programa, como comentarios, nombres significativos, indentación, descomponer el programa en módulos pequeños, etc... en la depuración ?.

- **Factores del programador**

¿ Qué importancia tiene el conocimiento del dominio del problema ?. Y, ¿ El conocimiento sintáctico del lenguaje de programación ?. Y, ¿ el conocimiento semántico del lenguaje de programación ?. Y, ¿ para que tipo de errores es crucial el entendimiento del programa ?.

- **Ayudas en la depuración**

Aunque las herramientas de depuración (debuggers), estaban disponibles para los sujetos en muchos experimentos, ¿ por qué la mayoría de los individuos no las usaban ?.

¿ Cuáles son las características más útiles de los depuradores para los programadores ?.

¿ Qué características de los depuradores son las más útiles para apoyar a una estrategia determinada de depuración ?.

A parte de los depuradores, ¿ qué otras herramientas del software son útiles para la depuración ?.

- **Modelo de depuración**

¿ Cómo podemos evaluar la validez del modelo de depuración propuesto en el capítulo 5 ?.

APENDICE A1

LISTADO DEL PROGRAMA DEFECTUOSO

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  El propósito de este programa es el de leer un    ;;;
;;;  conjunto de valores enteros,clasificarlos en orden;;;
;;;  ascendente y buscar unos ciertos valores-clave    ;;;
;;;  dentro del conjunto ordenado de valores.          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


program prueba(input,output,ficent);

const
     long   = 1000;
     numcla = 5 ;

type
     tipoarray = array [1..long] of integer;

var
     t : typoarray;
     i,conta,clave : integer;
     ficent : file of integer;


procedure LEERDATOS(var a:tipoarray;var conta:integer);

var
     indice : integer;

begin
     indice := 1;
     while not eof(ficent) do
          begin
            read(ficent,a[indice]);
          (*readln(ficent,a[indice]);*)
            indice := indice + 1;
          end;
     conta := indice;
   (*conta := indice - 1;*)
end;
```

```
procedure CLASIFICACION(var a:tipoarray;conta:integer);

var
     i,j,temp : integer;

begin
     for i := 1 to conta - 1 do
          for j := 0 to conta - 1 do
        (*for j := 1 to conta - 1 do*)
            if a[j] > a[j+1] then
               begin
                 temp := a[j];
                 a[j+1] := a[j];
                 a[j] := temp;
               end;
end;


function BUSQUEDA-binaria
               (a:tipoarray;clave,conta:integer):integer;
var
     inf,sup,mitad : integer;

begin
     inf := 1;
     sup := conta;
     while inf <> sup do
          begin
            mitad := (inf + sup) / 2;
          (*mitad := (inf + sup) div 2*)
            if clave >= a[mitad] then
          (*if clave <= a[mitad] then*)
              sup := mitad;
            else
              inf := mitad + 1;
          end;
     if clave = a[inf] then
          BUSQUEDA-binaria := inf;
     else
          BUSQUEDA-binaria := 0;
end;

begin (* PRINCIPAL *)
assign(ficent,'PRUEBA.DAT');
reset(ficent);
LEERDATOS(t,conta);
CLASIFICACION(t,conta);
for i := 1 to numcla do
     begin
          write('clave = ');
          readln(clave);
          writeln('clave = ',clave,'valor = ',
               BUSQUEDA-binaria(t,clave,conta);
     end;
end. (* PRINCIPAL *)
```

APENDICE A2

SALIDAS ESPERADAS

Clave = 4567	Valor = 19
Clave = 0	Valor = 3
Clave = -5	Valor = 2
Clave = 234	Valor = 0
Clave = 77	Valor = 8

APENDICE A3

TRABAJO PARCIAL DEL SUJETO N2

Tiempo	*Actividad*
2:40	- Ejecutar el programa. - Interpretar los mensajes de error visualizados por pantalla, obtenidos en el procedimiento LEERDATOS. - Examinar los mensajes de errores semánticos. - Recoger el protocolo verbal, sobre las especulaciones que hacen los sujetos, para corregir los errores semánticos. - Examinar el procedimiento LEERDATOS. - Examinar el procedimiento CLASIFICACION. - Examinar la función de BUSQUEDA-binaria. - Examinar el programa principal. - Examinar el procedimiento LEERDATOS. - Examinar el programa principal. - Examinar las declaraciones globales. - Ejecutar a mano LEERDATOS.

2:43 - Insertar var ficent:file of integer como un parámetro en la cabecera de LEERDATOS.
- Modificar LEERDATOS(t,conta) y poner LEERDATOS(t,conta,ficent) en el programa principal.
- Recoger el protocolo verbal habido, sobre las hipótesis, que los individuos creaban después de ejecutar el programa.
- Interpretar los mensajes de error visualizados del procedimiento LEERDATOS.
- Examinar los mensajes de error semánticos.
- Recoger el protocolo verbal de las especulaciones efectuadas,sobre los errores semánticos.
- Examinar el programa principal.
- Examinar el procedimiento LEERDATOS.
- Examinar el programa principal.
- Examinar el procedimiento LEERDATOS.
- Examinar el programa principal.
- Examinar LEERDATOS.

2:46 - Insertar writeln(indice) después de read,ficent,a[indice]) en el procedimiento LEERDATOS.
- Recoger el protocolo verbal de las hipótesis creadas por los sujetos después de ejecutar el programa.

- Visualizar los valores de índice en el programa.
- Interpretar los mensajes de errores semánticos habidos en el procedimiento LEERDATOS.
- Examinar los valores de índice.
- Examinar los mensajes de errores semánticos.
- Recoger el protocolo verbal de las especulaciones hechas para corregir los errores semánticos.
- Examinar la lista de datos de entrada.

2:48
- Quitar el writeln(indice) puesto en el procedimiento LEERDATOS.
- Examinar el procedimiento LEERDATOS.
- Ejecutar a mano el procedimiento LEERDATOS.
- Cambiar read(ficent,a[indice]) por readln(ficent,a[indice]).
- Recoger el protocolo verbal de las hipótesis efectuadas sobre la ejecución del programa.
- Interpretar los mensajes de errores semánticos producidos en el procedimiento CLASIFICACION y visualizados por pantalla.
- Examinar los mensajes de errores semánticos.

- Recoger el protocolo verbal de las especulaciones efectuadas por los sujetos, sobre los errores semánticos.
- Examinar el procedimiento CLASIFICACION.
- Examinar el programa principal.
- Examinar el procedimiento CLASIFICACION.

2:50 - Cambiar `if a[j] > a[j+1] then` por `if a[j] > a[j-1] then`

APENDICE B1

LISTADO DEL PROGRAMA DEFECTUOSO

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  El propósito de este programa es el de leer un    ;;;
;;;  conjunto de valores enteros,clasificarlos en orden;;;
;;;  ascendente y buscar unos ciertos valores-clave    ;;;
;;;  dentro del conjunto ordenado de valores.          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

program prueba2(input,output,ficent);

const
     long   = 1000;
     numcla = 5 ;

type
     tipoarray = array [1..long] of integer;

var
     t : typoarray;
     i,conta,clave,j,temp,indice : integer;
     inf,sup,mitad,indicear:integer;
     ficent : file of integer;

begin
     assign(ficent,'PRUEBA.DAT');
     reset(ficent);
     indice := 1;
     while not eof(ficent) do
          begin
            readln(ficent,a[indice]);
            indice := indice + 1;
          end;
     conta := indice;
     (*conta := indice - 1;*)
     for i := 1 to conta - 1 do
          for j := 1 to conta - 1 do
            if a[j] > a[j+1] then
               begin
                 temp := a[j];
                 (*temp := a[j+1]*)
                 a[j+1] := a[j];
                 a[j] := temp;
               end;
```

```
      for i := 1 to numcla do
           begin
             write('clave=');
             readln(clave);
             inf := 1;
             sup := conta;
             while inf <> sup do
               begin
                 mitad := (inf + sup) div 2;
                   if clave >= a[mitad] then
                   (*if clave <= a[mitad] then*)
                   sup := mitad;
                   else
                     inf := mitad + 1;
                   end;
                   if clave = a[inf] then
                   indicear := inf;
                   else
                   indicear := 0;
                   write('clave =',
                         clave,'valor=',indicear);
           end;
end. (* PRINCIPAL *)
```

APENDICE B2

SALIDAS ESPERADAS

Clave = 4567	Valor = 19
Clave = 0	Valor = 3
Clave = -5	Valor = 2
Clave = 234	Valor = 0
Clave = 77	Valor = 8

SALIDAS INCORRECTAS

Clave = 4567	Valor = 0
Clave = 0	Valor = 0
Clave = -5	Valor = 0
Clave = 234	Valor = 0
Clave = 77	Valor = 0

APENDICE B3

PROGRAMA PATRON

```
program prueba(input,output,ficent);

const
     long = 1000;
     numcla =5;

type
     tipoarr = array[1..long] of integer;

var
     t : tipoarr;
     i,j,temp,conta,indice,inf,sup,mitad : integer;
     indicear : integer;
     ficent : file of integer;

begin
     assign(ficent,'PRUEBA.DAT');
     reset(ficent);
end.
```

APENDICE B4

TEST PARA DESPUES DE LA SESION DE TRABAJO

- ¿ Qué tipos de clasificación (sort) y de búsqueda se usan en este programa ?. Rodea con un circulo la repuesta adecuada.

Sort : 1 Selección 2 Burbuja 3 Inserción.
Búsqueda : 1 Lineal 2 Binaria 3 Cuadrática.

- Si la instrucción 17 fuera cambiada a "read(ficent, a[indice]", ¿ qué cambios se producirían en la ejecución del programa ? (se especificó).

- Si la instrucción 23 fuera cambiada a "if a[j]<a[j+1]then", ¿ qué valores se deberían imprimir, para cada clave de entrada siguiente ?.

clave = 4567 valor = -------
clave = 77 valor = -------

- Si la instrucción 22 hubiera cambiado a" for j := conta -1 downto 1 do", ¿ cambiarían los resultados del programa ?. Si o no y explicar por qué.

- Si la instrucción 37 se cambiara a "mitad := (inf+sup) / 2;", ¿ qué ocurriría durante la ejecución del programa ? (se especificó).

- Si la instrucción 38 se cambiara a "if clave < <a[mitad]then", ¿ qué valores se imprimirían para las siguientes claves de entrada ?.

clave = 4567 valor = -------
clave = 670 valor = -------

- ¿ Para cada una de las siguientes claves de entrada qué valor se obtendría ?.

clave = 33 valor = -------
clave = 999 valor = -------

- Para obtener los siguientes valores ¿ qué valores clave de entrada habría que introducir ?.

valor = 12 clave = -------
valor = 0 clave = -------

- ¿ qué habría que cambiar en el programa, para poder introducir 7 valores clave ?.

APENDICE B5

PROGRAMA PARA EL RELLENADO DE HUECOS

```
01 program prueba3(input,output,ficent);

02 const

03    long    = 1000;
04    numcla  = 5 ;

05 type

06    tipoarray = array [1..long] of integer;

07 var

08    t : typoarray;
09    i,conta,clave,j,temp,indice,
      inf,sup,mitad,indicear:integer;
10    ficent : file of integer;

11 begin

12    assign(ficent,'PRUEBA.DAT');
13    reset(ficent);
14    indice := 1;
15    while --------------- do
16          begin
17            readln(ficent,a[indice]);
18            indice := ----------;
18          end;
20    conta := ---------;
21    for i := 1 to ------------- do
22          for j := 1 to ------------ do
23            if ---------------- then
24               begin
25                 temp := a[j+1];
26                 a[j+1] := a[j];
```

```
27                a[j] := temp;
28              end;



29    for i := 1 to numcla do
30          begin
31            write('clave=');
32            readln(clave);
33            inf := 1;
34            sup := conta;
35            while ----------------- do
36              begin
37                mitad := (inf + sup) -------------;
38                  if ---------------------- then
39                    sup := mitad;
40                  else
41                    inf := ---------- ;
42                  end;
43                  if ----------------- then
44                    indicear := inf;
45                  else
46                    indicear := 0;
47                  write('clave =',
                          clave,'valor =',indicear);
48          end;
49 end. (* PRINCIPAL *)
```

BIBLIOGRAFIA

[ATWO78] ATWOOD, M.E.; RAMSEY, H.R.. ***Cognitive Structures in the Comprehension and Memory of Computer Programs: An Investigation of Computer Program Debugging.*** TR-78-A21, U.S. Army Research Institute for the Behavioral and Social Sciences, Alexandria, Virginia, August, 1978.

[BARR90] BARREDO, M.A.. ***Metrica de la Medida.*** Universidad de Deusto, Bilbao, 1990.

[BOEH81] BOEHM, B.W.. ***Software Engineering Economics.*** Prentice-Hall, Inc., 1981.

[BROO80] BROOKS, R.E.. ***Studying Programmer Behavior Experimentally: The Problem of Proper Methodology.*** Communications of the ACM, Volume 23, Number 4, Abril 1980, pp. 207-213.

[BROW73] BROWN, A.R.; SAMPSON, W.A.. ***Program Debugging.*** American Elseveir, 1973.

[CARV89] CARVER, D.. ***Programmer variation in software debugging approaches.*** International Journal of Man-Machine Studies, 31,3, Sep. 1989, pp. 315-322.

[COOK84] COOK, C.R.; BREGAR, W.S.; FOOTE, D.A.. ***Preliminary Investigation of the Use of the Cloze Procedure as a Mesure of Program Understanding.*** Information Processing & Management, Vol, 20, No. 1-2, 1984, pp. 199-208.

[CURT79] CURTIS, B.. ***In Search of Software Complexity.*** Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost. Washington, D.C., IEEE Computer Society, 1979, pp. 95-206.

[ERIC80] ERICSSON, K.A.; SIMON, H.A.. ***Verbal Reports as Data.*** Psychological Review, Volume 87, 1980. pp 215-251.

[FINN87] FINNIE, G.. ***Some experimental result from non-procedural languages.*** International Journal of Man Machine Studies, 25, 5, November, 1987, pp. 469-478.

[GANN77] GANNON, J.D.. ***An Experimental Evaluation of Data Type Conventions.*** Comunications of the ACM, 1977, Volume 20, Number 8, pp. 584-595.

[GLAS82] GLASS, R.L.. ***Modern Programming Practices: A Report from Industry.*** Prentice-Hall, Inc., 1982.

[GOUL75] GOULD, J.D.. ***Some Psychological Evidence on How People Debug Computer Programs.*** International Journal of Man-Machine Studies, 7, 1975, pp. 151-182.

[GOUL89] GOULD, J.D.. ***Conferences proceedings of human factors in computing system.*** Proceedings of the ACM, Siggraph, 1989.

[GUGE86] GUGERTY, L.; OLSON, G.M.. ***Comprehension differences in Debugging by Skilled and Novice Programmers.*** Empirical Studies of Programmers, Soloway, E. and Iyengar, S. (Eds.). Ablex Inc., Norwood, New Jersey, 1986, pp. 13-27.

[HALS77] HALSTEAD, M.. ***Elements of Software Science.*** Elsevier, 1977.

[KESS86] KESSLER, C.M.; ANDERSON, J.R.. ***A Model of Novice Debugging in LISP.*** Empirical Studies of Programmers, Soloway, E. and Iyengar, S. (Eds.). Ablex Inc., Norwood, New Jersey, 1986, pp. 198-212.

[LUKE80] LUKEY, F.J.. ***Understanding and Debugging Programs.*** International Journal of Man Machine Studies 12, 2, 1980, pp. 189-202.

[MCCA76] MCCABE, T.. ***A Complexyty Measure.*** IEEE Transactions of Software Engineering, December 1976, pp. 308-320.

[MOHE82] MOHER, T.; SCHNEIDER, G.M.. ***Methodology and Experimental Research in Software Engineering.*** International Journal of Man Machine Studies, 16, 1982, pp.65-87.

[MYER79] MYERS, G.J.. ***The Art of Software Testing.*** John Wiley and Sons, 1979.

[MYER78] MYERS, G.J.. ***A Controlled Experiment in Programming Testing and Code Walkthroughs / Inspection.*** Communications of the ACM, 1978, Volume 21, Number 9, pp. 760-768.

[OMAN88] OMAN, P.; COOK,C.R.; NANJA, M.. ***Effects of Programming Experience in Debugging Semantic Errors.*** Sigplan Notices 23,12, December 1988 pp. 69-78.

[OTT 89] OTT, L.; THUSS, J.. ***Software Engineering.*** Proceeding of the Eleventh International Conference. ACM Press, New York, May 1989, pp. 406.

[PARR89] PARRINGTON, N.; ROPER, M.. ***Understanding software testing.*** Halsted Press, New York, 1989.

[SACK68] SACKMAN, H.; ERICKSON, W.; GRANT,E.. ***Exploratory and Experimental Studies Comparing on-line and off-line Programming Performance.*** Comunications of the ACM, Volume 1, 1968, pp.3-11.

[SHEP79] SHEPPARD, S.B.; CURTIS, B.; MILLIMAN, P.; LOVE, T.. ***Modern Coding Practices and Programmer Performance.*** Computer, Volume 12, Number 12, 1979, pp. 41-49.

[SHNE77] SHNEIDERMAN, B.; MAYER, R.; MACKAY, D.; HELLER, P.. *Experimental Investigations of the Utility of Detailed Flowcharts in Programming.* Communications of the ACM, Volume 20,Number 6, June 1977, pp. 373-381.

[TASS78] TASSEL, D.V.. *Program Style, Design, Efficiency, Debugging, and Testing.* Prentice Hall, Inc., 1978.

[VESS85] VESSEY, I.. *Expertise in Debugging Computer Programs: A Process Analysis.* International Journal of Man Machine Studies, Volume 23, 1985, pp. 459-494.

[WARD86] WARD,R.. *Debugging C.* Que Corporation, 1986.

[WEIS82] WEISER, M.. *Programmers use Slices when Debugging.* Communications of the ACM, Volume 25, Number 7, July 1982, pp. 446-452.

[WEIS86] WEISER, M.; LYLE, J.. *Experiments of Slicing Based Debugging Aids.* Empirical Studies of Programmers, Soloway, E. and Iyengar, S. (Eds.). Ablex Inc., Norwood, New Jersey, 1986, pp. 187-197.

[WIED86] WIEDENBECK, S.. ***Processes in Computer Program Comprehension.*** Empirical Studies of Programmers, Soloway, E. and Iyengar, S. (Eds.). Ablex Inc., Norwwod, New Jersey, 1986, pp. 48-57.

[YOUN74] YOUNGS, E.. ***Human Errors in Programming.*** International Journal of Man Machine Studies, Volume 6, 1974, pp. 361-376.

[ZELK80] ZELKOWITZ, M.V.; SHAW, A.C.; GANNON, J.D.. ***Principles of Software Engineerig and Design.*** Prentice-Hall, Inc., 1980.

[ZWEB89] ZWEBEN, S.. ***Software Engineering Education Set Conference.*** Sibbis, N. (Ed). 1989.