# FACULTAD DE INGENIERÍA
# UNIVERSIDAD DE LA REPÚBLICA

# Un Enfoque Comparativo de las Técnicas de Verificación Todos los Usos y Cubrimiento de Sentencias

## Proyecto de Grado

**Carmen Bogado**
cmbogado@gmail.com

**Silvana Moreno**
silvanamoren@gmail.com

**Tutor**
Diego Vallespir
dvallesp@fing.edu.uy

**Montevideo, Uruguay**
**Diciembre, 2009**

# Resumen

Si bien las pruebas unitarias ya están fuertemente establecidas en la industria de software aún no se conoce qué tan efectiva es cada técnica de verificación, ni el costo asociado a su aplicación. Obtener dicho conocimiento no es sencillo porque el comportamiento puede variar dependiendo de la persona que aplica la técnica, del lenguaje de programación y del tipo de aplicación, entre otros. Además, no es posible demostrar formalmente cuál es la efectividad y el costo de una técnica de verificación. Una opción para obtener información es mediante la realización de experimentos formales.

Uno de los principales objetivos de este proyecto de grado es obtener información que permita comparar la efectividad y costo de las técnicas de verificación Cubrimiento de Sentencias (CS) y Todos los Usos (TU). Para obtener esta información diseñamos y ejecutamos un experimento formal.

También, desarrollamos un marco de comparación que provee formalidad en la comparación de experimentos teniendo en cuenta sus características más relevantes. Se utiliza el marco para comparar algunos experimentos formales en Ingeniería de Software. Al estudiar y comparar los experimentos se logra un mayor conocimiento para definir nuestro experimento formal.

Los sujetos que participan en el experimento son estudiantes de la carrera Ingeniería en Computación de la Facultad de Ingeniería de la Universidad de la República, y participan del mismo en el marco de una asignatura. Estos son entrenados mediante clases teórico-prácticas donde se explica como aplicar las técnicas y usar los materiales de soporte. Las clases son preparadas y dictadas por las autoras de este proyecto de grado bajo la supervisión del tutor.

Los resultados obtenidos aplicando pruebas de hipótesis indican que no existe evidencia estadística para afirmar que una técnica es más efectiva que la otra. Sin embargo, sí existe evidencia estadística que indica que TU es más costosa que CS.

**Palabras clave:** Ingeniería de Software, Experimento Formal, Técnica de Verificación, Pruebas Unitarias.

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 1

# Introducción

La construcción de software de calidad se ha convertido en un factor clave de éxito en el desarrollo de numerosos productos y servicios en todos los sectores de la industria. Sin embargo, uno de los problemas planteados en el área Ingeniería de Software es el costo y efectividad de las distintas técnicas de verificación de software.

## 1.1. Motivación

Normalmente se usa una técnica de prueba de software para verificar una unidad de software. Sin embargo, no se sabe cuál elegir y tampoco si diferentes técnicas se comportan igual. Si bien las pruebas unitarias ya están fuertemente establecidas en la industria de software aún no se conoce qué tan efectiva es cada técnica de verificación. Tampoco se conoce el costo asociado o si la efectividad varía según los distintos tipos de defecto. En definitiva, al momento de realizar pruebas unitarias la decisión acerca de cuál o cuáles técnicas usar no es trivial.

Obtener el conocimiento de cómo se comportan las técnicas no es sencillo porque el mismo puede variar dependiendo de la persona que aplica la técnica, del lenguaje de programación y del tipo de aplicación que se prueba (sistema de información, robótica, etc.), entre otros. Además, no es posible demostrar formalmente cuál es la efectividad y el costo de una técnica de verificación.

Una opción para obtener información acerca de la efectividad y el costo es mediante la realización de experimentos formales. Algunos avances se han realizado pero aún queda un largo camino por recorrer. Existen diversos experimentos formales para conocer cómo se comportan distintas técnicas de pruebas de software. El primero del cual tenemos conocimiento data de 1978 [15].

Han pasado varios años de investigación empírica en el tema y, sin embargo, no se han logrado aún resultados definitivos. En *A look at 25 years of data* los autores examinan diferentes experimentos de pruebas de software llegando también a esta conclusión [14].

En dicho artículo también se menciona la dificultad que existe para poder comparar diferentes experimentos pero no se presenta una solución para esto. Algunos de los problemas detectados en los programas son: distintos tamaños, diferentes lenguajes en los que están codificados y documentación insuficiente. Además, las técnicas utilizadas se encuentran parcialmente definidas. Contar con

un marco de comparación de experimentos es necesario para poder compararlos formalmente y para poder avanzar en la construcción de nuevos experimentos basándose en experimentos anteriores.

## 1.2. Objetivos

Debido a que aún no se conoce con certeza el comportamiento de las técnicas de verificación, en el Grupo de Ingeniería de Software de la Facultad de Ingeniería se están llevando adelante varios experimentos formales para obtener datos más precisos sobre **la efectividad y el costo de técnicas de verificación**. Este proyecto de grado forma parte del trabajo realizado en el grupo [21], [19], [22], [20], [6], [3], [23], [1].

Los objetivos de este proyecto de grado son:

- Construir un marco de comparación de experimentos formales que buscan comparar el comportamiento de diferentes técnicas de verificación.
- Comparar determinados experimentos conocidos que evalúan técnicas de verificación.
- Realizar un experimento formal para comparar la efectividad y el costo de las técnicas de verificación Cubrimiento de Sentencias (CS) y Todos los Usos (TU).

Se pretende entonces la construcción de un marco de comparación original que permita comparar experimentos a través de sus características más relevantes. Se busca realizar un análisis comparativo utilizando el marco presentado con determinados experimentos conocidos, realizados por Basili y Selby[2] (B-S) , Kamsties y Lott[11] (K-L), Macdonald y Miller[12] (M-M), y Juristo y Vega[10] (J-V).

Dado que la efectividad y el costo no son atributos cuyo valor pueda ser demostrado formalmente, es necesario exponer evidencia empírica obtenida mediante la realización de experimentos. Para esto, se pretende llevar a cabo un experimento formal que permita obtener información acerca de la efectividad y el costo de las técnicas de verificación unitaria CS y TU.

Se define la efectividad a medir en el experimento formal de la siguiente forma:

$Efectividad = (\frac{\#TotalDefectosDetectados}{\#EstimadoDefectosTotales}) * 100$

El costo de aplicar cada técnica se calcula como la sumatoria de los siguientes datos:

- **Tiempo de Diseño de los Casos de Prueba (TDCP)**: Tiempo que se invierte en el diseño y codificación en JUnit (ver 1.3.1) de los casos de prueba que cumplen con la técnica.
- **Tiempo de Ejecución de los Casos de Prueba (TE)**: Corresponde al tiempo de ejecución de los casos de prueba que requieren interacción con el usuario. Para los casos de prueba codificados en JUnit no se registra el tiempo de ejecución ya que se considera nulo.

Por lo tanto, el costo de las técnicas CS y TU se calcula de la siguiente forma:

$Costo(CS) = TDCP_{CS} + TE_{CS}$
$Costo(TU) = TDCP_{TU} + TE_{TU}$

## 1.3. Conceptos Teóricos

En el desarrollo de software, durante y después del proceso de implementación, el programa que se está desarrollando debe ser probado para asegurar que satisface su especificación y brinda la funcionalidad esperada. La verificación tiene lugar en cada etapa del proceso de software, incluyendo revisiones a los requerimientos, al diseño, inspecciones de código, pruebas unitarias, pruebas de integración y pruebas al producto final. La **verificación de software** responde a la pregunta: ¿Estamos construyendo el producto correctamente? Se comprueba que lo que se construye cumple los requisitos funcionales y no funcionales de su especificación.

Dentro del proceso de desarrollo de software existen las pruebas de software para comprobar los sistemas. Las **pruebas de software** implican ejecutar parte del software con datos de pruebas. Se examinan los resultados obtenidos en la ejecución del software para comprobar que funcione tal y como se requiere.

Las pruebas de software ejecutan un programa con la intención de provocar fallas para luego poder encontrar las faltas. Una **falla** ocurre cuando un programa no se comporta de manera adecuada. La falla es una propiedad de un sistema en ejecución.

Las fallas se manifiestan debido a que el código del programa contiene una **falta**. Se utiliza *defecto* como sinónimo de falta. Las faltas pueden ser inyectadas en el código, lo que significa que se introducen en el programa de forma intencional. Esta práctica es muy utilizada en la ejecución de experimentos formales.

A su vez, las faltas (que no son inyectadas) son causadas por un **error**. Un **error** es una acción humana que provoca que el software contenga una falta. Dicha falta puede generar una falla en el sistema.

Existen dos enfoques de pruebas, las pruebas de caja blanca y las pruebas de caja negra. Se dice que una prueba es de **caja negra** cuando prescinde de los detalles del código y se limita a lo que se ve desde el exterior para probar el software. Por oposición al término caja negra se denomina **caja blanca** al caso contrario. En este caso sí se usa el código para generar los casos que prueban el software.

Las técnicas de caja blanca se pueden clasificar en las basadas en el flujo de control del programa y las basadas en el flujo de datos del programa.

- Las técnicas basadas en el **flujo de control** del programa expresan los cubrimientos del testing en términos del grafo de flujo de control del programa.

- Las técnicas basadas en el **flujo de datos** del programa expresan los cubrimientos del testing en términos de las asociaciones definición-uso del programa.

### 1.3.1. JUnit

JUnit[1] es un conjunto de bibliotecas que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. JUnit permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el resultado obtenido en base al resultado esperado. Si la clase cumple con la especificación entonces JUnit indicará que el método pasó exitosamente la prueba, en caso de que el resultado esperado sea diferente al obtenido JUnit indicará un fallo en el método correspondiente.

En la actualidad las herramientas de desarrollo como NetBeans y Eclipse cuentan con *plug-ins* que permiten la generación de las plantillas necesarias para la creación de las pruebas de una clase Java en forma automática, permitiendo al programador enfocarse en la prueba y el resultado esperado.

## 1.4. Propuesta y Realización

Para cumplir con los objetivos planteados fue indispensable adquirir conocimientos acerca de los fundamentos de la Ingeniería de Software Empírica. Este conocimiento nos brindó información acerca de la realización de experimentos formales en esta área en particular, y nos permitió diseñar y ejecutar nuestro experimento. Para ello se realizó un estudio sobre Ingeniería de Software Empírica, abarcando conceptos básicos de experimentación en el área, fases que componen a los experimentos (objetivos, diseño, operación y análisis de datos), y terminología (unidad experimental, parámetros, factores, alternativas, variables de respuesta, entre otros)[9].

En este trabajo se comparan experimentos formales que evalúan técnicas de verificación, y se define y ejecuta un experimento formal.

Para comparar los experimentos desarrollamos un marco original que brinda formalidad en la comparación de experimentos. Se utiliza el marco para comparar las características más relevantes de los experimentos formales realizados por B-S, K-L, M-M y J-V. Al estudiar y comparar los experimentos se logró un mayor conocimiento para definir nuestro experimento formal.

El experimento formal que se definió y ejecutó busca comparar la efectividad y el costo de las técnicas CS y TU. Se consultó la literatura para obtener una definición apropiada de estas técnicas a aplicar en el experimento.

El diseño del experimento es de un factor (técnica) con dos alternativas (CS y TU). Consta de una única unidad experimental: el programa a probar. Las variables de respuesta consideradas son la efectividad y el costo de las técnicas.

Los sujetos que participaron en el experimento son estudiantes de la carrera Ingeniería en Computación de la Facultad de Ingeniería de la Universidad de la República, y participaron del mismo en el marco de una asignatura.

Para llevar a cabo la ejecución del experimento los sujetos fueron entrenados durante las denominadas experiencias I y II (o experiencias iniciales) mediante clases teórico-prácticas. Las clases fueron preparadas y dictadas por las autoras de este proyecto de grado. La duración de cada experiencia fue de aproximadamente 3 semanas y en total participaron 21 sujetos.

[1] www.junit.org

En la clase teórica se explicaron las técnicas a aplicar y los materiales de soporte.

La clase práctica consistió en la aplicación de las técnicas de verificación CS y TU por parte de los sujetos a un simple programa escrito en Java.

La experiencia I fue muy útil para adquirir información valiosa sobre la adecuación de la capacitación brindada a los estudiantes, y permitió mejorarla para la experiencia II. Además, otorgó una visión general de las dificultades afrontadas en la aplicación de las técnicas.

La ejecución del experimento se realizó en una única instancia con una duración de 8 semanas. Participó un subconjunto de los sujetos que participaron en las experiencias iniciales. De los 14 sujetos que realizan el experimento, 6 aplicaron CS y 8 aplicaron TU.

Se aplicaron pruebas de hipótesis a los datos recolectados en el experimento. Los resultados obtenidos indicaron que no existe evidencia estadística para afirmar que una técnica es más efectiva que la otra. Al comparar el costo de las técnicas se observó que existe evidencia estadística que indica que TU es más costosa que CS.

Los resultados obtenidos podrían estar condicionados al programa utilizado, a los tipos de defectos que contiene, la cantidad de sujetos que ejecutan el experimento, entre otros. Se necesita realizar más experimentos que posean mayor cantidad de observaciones para obtener resultados más significativos respecto a la efectividad y el costo de distintas técnicas de verificación.

## 1.5. Publicaciones

En el marco de este proyecto se publica el artículo:

- Vallespir, D., Moreno, S., Bogado, C., Herbert, J.: *Towards a framework to compare formal experiments that evaluate verification techniques*, Proceedings of the X Mexican International Conference in Computer Science, Ciudad de México, México, 2009.

## 1.6. Estructura del Documento

Este documento se estructura en 7 capítulos además del presente.

El capítulo **Conceptos de Ingeniería de Software Empírica** presenta un resumen de lo que significa realizar experimentos en Ingeniería de Software, y brinda una breve introducción a los distintos tipos de experimentos que existen, profundizando en el tipo que se va a utilizar para este estudio. También se define parte de la terminología que se va a usar en el resto del documento.

En el capítulo **Marco de Comparación de Experimentos** se desarrolla un marco de comparación de experimentos formales que evalúan técnicas de verificación. Se presentan los experimentos realizados por Basili-Selby, Kamstries-Lott, Macdonald-Miller y Juristo-Vega; y se utiliza el marco propuesto para compararlos.

En el capítulo **Cubrimiento de Sentencias y Todos los Usos** se detallan las técnicas de verificación aplicadas en el experimento realizado. Se ilustra su aplicación con algunos ejemplos.

En el capítulo **Introducción al Experimento Formal** se presenta una introducción al experimento realizado. Se describen las fases que componen el experimento y el conjunto de artefactos necesario para brindar soporte al mismo.

La capacitación brindada a los sujetos para llevar a cabo el experimento se presenta en el capítulo **Experiencias Iniciales**.

En el capítulo **Experimento Formal** se detalla el experimento realizado. En este capítulo se presenta el diseño elegido, la operación, el análisis y la evaluación de los datos recolectados, entre otros.

Por último, el capítulo **Conclusiones y Trabajos a Futuro** presenta las conclusiones acerca de este trabajo, y tareas que resultan interesantes para abordar en futuros trabajos.

Se incluyen 3 anexos que contienen información complementaria.

En el **Anexo A** se presentan las **clases de capacitación** brindadas a los sujetos, donde se explica la aplicación de las técnicas y el uso de la planilla de registro de tiempos y defectos.

El **Anexo B** contiene la **Guía de Verificación** que especifica el proceso de verificación seguido por los sujetos.

El **Anexo C** contiene las respuestas de los sujetos a la **encuesta** realizada para obtener su visión respecto al curso.

Adicionalmente, se entregan impresos por separado los dos artículos escritos junto con el tutor durante el proyecto de grado.

# Capítulo 2

# Conceptos de Ingeniería de Software Empírica

En este capítulo se presenta un Reporte Técnico perteneciente al InCo-PEDECIBA. Este reporte expone conceptos de Ingeniería de Software Empírica y ha sido generado para ser utilizado en proyectos de grado desarrollados en el área.

Los autores del reporte son: Cecilia Apa, Rosana Robaina, Stephanie de León, Diego Vallespir.

## 2.1. Introducción

El Grupo de Ingeniería de Software (GrIS) del Instituto de Computación, Facultad de Ingeniería, Universidad de la República se encuentra realizando experimentos formales para conocer el comportamiento de distintas técnicas de verificación. Además, hace varios años que se realizan pruebas de procesos de desarrollo de software en el marco de una asignatura llamada Proyecto de Ingeniería de Software [18]. Si bien estas pruebas no son formales, es interesante en un futuro formalizarlas.

Para poder realizar experimentos formales se deben conocer los conceptos, las técnicas y las herramientas normalmente usadas en la Ingeniería de Software Empírica (ISE). Esta área, relativamente nueva de la Ingeniería de Software (IS), ha causado un impacto considerable en la comunidad científica y en la industria, teniendo su propia revista internacional (Empirical Software Engineering: An International Journal)[1] desde el año 1996.

Este reporte tiene como objetivo presentar los conceptos fundamentales de ISE. Se pretende que este documento sea utilizado por Proyectos de Grado de la carrera Ingeniería en Computación que se encuentran realizando trabajos de ISE con el GrIS. Distintos estudiantes de Proyecto de Grado se encuentran trabajando con nosotros en estos temas y parece razonable tener un documento que sea común a todos estos proyectos. De esta manera los estudiantes pueden usar este documento como punto de partida para comprender la ISE. Además,

[1] http://www.springer.com/computer/programming/journal/10664

pueden incluir este documento como parte de su informe de proyecto evitando tener un enfoque distinto de la ISE en cada Proyecto de Grado.

Este reporte se basa casi completamente en los libros *Experimentation in Software Engineering: An Introduction* [24], *Basics of Software Engineering Experimentation* [9] y *Software Metrics - A Rigorous And Practical Approach* [4].

En la sección 2.2 se presentan los distintos enfoques y estrategias de la ISE. Una de estas estrategias es la de experimentos formales, estos se describen en la sección 2.3. Por último, la sección 2.4 describe un proceso para llevar adelante un experimento formal.

## 2.2. Enfoques y Estrategias

La ISE utiliza métodos y técnicas experimentales como instrumentos para la investigación. La evidencia empírica proporciona un soporte para la evaluación y validación de atributos (p.e. costo, eficiencia, calidad) en varios tipos de elementos de Ingeniería de Software (p.e. productos, procesos, técnicas, etc.). Se basa en la experimentación como método para corresponder ideas o teorías con la realidad, la cual refiere a mostrar con hechos las especulaciones, suposiciones y creencias sobre la construcción de software.

Se pueden distinguir dos enfoques diferentes al realizar una investigación empírica: el enfoque cualitativo y el cuantitativo. El enfoque **cualitativo** se basa en estudiar la naturaleza del objeto y en interpretar un fenómeno a partir de la concepción que las personas tienen del mismo. Los datos que se obtienen de estas investigaciones están principalmente compuestos por texto, gráficas e imágenes, entre otros.

El enfoque **cuantitativo** se corresponde con encontrar una relación numérica entre dos o más grupos. Se basa en cuantificar una relación o comparar variables o alternativas bajo estudio. Los datos que se obtienen en este tipo de estudios son siempre valores numéricos, lo que permite realizar comparaciones y análisis estadístico.

Es posible utilizar los enfoques cualitativos y cuantitativos para investigar el mismo tema, pero cada enfoque responde a diferentes interrogantes. Se puede considerar que estos enfoques son complementarios más que competitivos, ya que el enfoque cualitativo puede ser usado como base para definir la hipótesis que luego puede ser correspondida cuantitativamente con la realidad. Cabe destacar que las investigaciones cuantitativas pueden obtener resultados más justificables y formales que los cualitativos.

Hay 3 tipos principales de técnicas o estrategias para la investigación empírica: las encuestas, los casos de estudio y los experimentos.

Las **encuestas** se utilizan, o bien cuando una técnica o herramienta ya ha sido usada, o antes de comenzar a hacerlo. Son estudios retrospectivos de las relaciones y los resultados de una situación. Las encuestas son realizadas sobre una muestra representativa de la población, y luego los resultados son generalizados al resto de la población. El ámbito donde son más usadas es en ciencias sociales, por ejemplo, para determinar cómo la población va a votar en la siguiente elección.

En la Ingeniería de Software Empírica las encuestas se utilizan de forma similar, se obtiene un conjunto de datos de un evento que ha ocurrido para determinar cómo reacciona la población frente a una técnica, herramienta o

método particular, o para determinar relaciones o tendencias. En un estudio es fundamental seleccionar correctamente las variables a estudiar, pues de ellas dependen los resultados que se pueden obtener. Si los resultados no permiten concluir sobre los objetivos del estudio se han elegido mal las variables.

Una de las características más relevantes de las encuestas es que proveen un gran número de variables para estudiar. Esto hace posible construir una variedad de modelos y luego seleccionar el que mejor se ajusta a los propósitos de la investigación, evitando tener que especular cuáles son las variables más relevantes. Dependiendo del diseño de la investigación (cuestionario) las encuestas pueden ser clasificadas como cualitativas o cuantitativas.

Los **casos de estudio** son métodos observacionales, se basan en la observación de una actividad o proyecto durante su curso. Son utilizados para monitorear proyectos, o actividades y para investigar entidades o fenómenos en un período específico.

En un caso de estudio se identifican los factores clave que pueden afectar la salida de una actividad, y se documentan las entradas, las limitaciones, los recursos y las salidas. El nivel de control de la ejecución es menor en los casos de estudio que en los experimentos. Esto se debe principalmente a que en los casos de estudio no se controla, sólo se observa, contrario a lo que ocurre en los experimentos.

Los casos de estudio son muy útiles en el área de Ingeniería de Software, se usan en la evaluación industrial de métodos y herramientas. Además, son fáciles de planificar aunque los resultados son difíciles de generalizar y comprender. Los casos de estudio no manipulan las variables, sino que éstas son determinadas por la situación que se está investigando.

Al igual que las encuestas, los casos de estudio pueden ser clasificados como cualitativos o cuantitativos dependiendo de lo que se quiera investigar del proyecto en curso.

Los **experimentos** son generalmente ejecutados en un ambiente de laboratorio, el cual brinda un alto grado de control. El objetivo en un experimento es manipular una o más variables y controlar el resto. Un experimento es una técnica formal, rigurosa y controlada de llevar a cabo una investigación.

En las secciones siguientes se profundiza en los experimentos formales como técnica de investigación.

## 2.3. Experimentos Formales

Como se mencionó anteriormente, los experimentos son una técnica de investigación en la cual se quiere tener un mejor control del estudio y del entorno en el que éste se lleva a cabo.

Los experimentos son apropiados para investigar distintos aspectos de la IS, como ser: confirmar teorías, explorar relaciones, evaluar la exactitud de los modelos y validar medidas. Tienen un alto costo respecto de las otras técnicas de investigación, pero a cambio ofrecen un control total de la ejecución y son de fácil replicación.

### 2.3.1. Terminología

En esta sección se presentan los términos más comunmente usados en diseño experimental. Se usan dos ejemplos de experimentos a lo largo de esta sección para introducir dichos términos.

En el primer ejemplo se tiene un experimento en el campo de la medicina, mediante el cual se quiere conocer la efectividad de los analgésicos en las personas entre 20 y 40 años de edad, llamado "Efec-Analgésicos".

En el segundo ejemplo, se quiere conocer la efectividad de 5 técnicas de verificación sobre un conjunto de programas, llamado "Efec-Técnicas".

Los objetos sobre los cuales se ejecuta el experimento son llamados **Unidades Experimentales** u objetos experimentales. La unidad experimental en un experimento de Ingeniería de Software podría llegar a ser el proyecto de software como un todo o cualquier producto intermedio durante el proceso.

Para *Efec-Analgésicos* se tiene que la unidad experimental es un grupo de personas entre 20 y 40 años de edad, en ese grupo de personas es en donde se observa el efecto de los analgésicos. En el ejemplo de *Efec-Técnicas*, se tiene que la unidad experimental es el conjunto de programas sobre los cuales se aplican las técnicas de verificación.

Aquellas personas que aplican los métodos o técnicas a las unidades experimentales se les llama **Sujetos Experimentales**. A diferencia de otras disciplinas, en la IS los sujetos experimentales tienen un importante efecto en los resultados del experimento, por lo tanto es una variable que debe ser cuidadosamente considerada.

En *Efec-Analgésicos* los sujetos son aquellas personas que administran los analgésicos a ser consumidos por los pacientes (enfermeros por ejemplo). Cómo los enfermeros administran los analgésicos a los pacientes no es algo que se espere vaya a afectar el experimento. La forma en que un enfermero administra un analgésico a un paciente es poco probable que sea diferente a la de otro, y aunque lo fuera, no se espera que afecte los resultados del experimento.

En *Efec-Técnicas* los sujetos pueden ser ingenieros que aplican la técnica en un conjunto particular de programas (unidad experimental). En este caso, los resultados del experimento podrían diferir mucho de acuerdo a la formación y experiencia de los ingenieros, así como también la forma en que las técnicas son aplicadas, incluso el estado de ánimo del verificador podría influir en los resultados.

El resultado de un experimento es llamado **Variable de Respuesta**. Este resultado debe ser cuantitativo. Una variable de respuesta puede ser cualquier característica de un proyecto, fase, producto o recurso que es medida para verificar los efectos de las variaciones que se provocan de una aplicación a otra. En ocasiones, a una variable de respuesta se le llama también variable dependiente.

En *Efec-Analgésicos* la efectividad podría ser medida en el grado de alivio del dolor en un determinado lapso de tiempo, o bien qué tan rápido el analgésico alivia el dolor. En ambos casos, la variable debe ser expresada cuantitativamente. En el primer caso se podría tener una escala, en la cual cada valor signifique un grado de alivio del dolor, en el segundo caso, el lapso de tiempo en que el analgésico es efectivo, se podría medir en minutos.

Para *Efec-Técnicas* la efectividad podría ser medida de acuerdo a la cantidad de defectos que encuentra la técnica sobre la cantidad de defectos totales del software verificado.

Un **Parámetro** es cualquier característica que permanezca invariable a lo largo del experimento. Son características que no influyen o que no se desea que influyan en el resultado del experimento o en la variable de respuesta. Los resultados del experimento serán particulares a las condiciones definidas por los parámetros. El conocimiento resultante podrá ser generalizado solamente considerando los parámetros como variables en sucesivos experimentos y estudiando su impacto en las variables de respuesta.

En el ejemplo de *Efec-Analgésicos* se tiene que el rango de edades (entre 20 y 40 años de edad) es un parámetro del experimento, los resultados serán particulares para el rango establecido.

En *Efec-Técnicas* un parámetro posible es el tamaño del software a ser verificado (por ejemplo: que tenga entre 200 y 500 LOCs). Otro parámetro para este experimento podría ser la experiencia de los verificadores, en este caso se podría fijar la experiencia en un determinado nivel.

Cada característica del desarrollo de software a ser estudiada que afecta a las variables de respuesta se denomina **Factor**. Cada factor tiene varias alternativas posibles. Lo que se estudia, es la influencia de las alternativas en los valores de las variables de respuesta. Los factores de un experimento son cualquier característica que es intencionalmente modificada durante el experimento y que afecta su resultado.

El factor en *Efec-Analgésicos* es "los analgésicos", en *Efec-Técnicas* tenemos que el factor es "las técnicas de verificación". Para ambos casos el factor se varía intencionalmente (se varía el tipo de analgésico o tipo de técnica de verificación) para ver cómo afecta en la efectividad.

Los posibles valores de los factores en cada unidad experimental son llamados **Alternativas** o niveles. En algunos casos también se les llama tratamientos.

Las alternativas de *Efec-Analgésicos* son los distintos tipos de analgésicos que se estudian en el experimento (p.e. Aspirina, Zolben, etc). De igual forma, para *Efec-Técnicas* las distintas alternativas son los 5 tipos distintos de técnicas que se estudian.

El intento de ajustar determinadas características de un experimento a un valor constante no es siempre posible. Es inevitable y a veces indeseable tener variaciones de un experimento a otro. Éstas variaciones son conocidas como **Bloqueo de Variables** y dan lugar a un determinado tipo de diseño experimental, llamado *block design*.

Una variable indeseada para *Efec-Analgésicos* podría ser el "umbral del dolor". Si se aplica una alternativa de analgésico a personas con umbral del dolor alto y otra alternativa a personas con umbral del dolor bajo, se tendría una variación indeseada, ya que la efectividad que se mida de los distintos tipos de analgésico va a variar no solamente por el tipo de analgésico administrado sino por el nivel de umbral del dolor del paciente al cual se lo administra.

En el caso de *Efec-Técnicas*, podría resultar que la experiencia de los verificadores resultase una variación indeseada si no se la tiene en cuenta previamente. Una forma de bloquear la experiencia en verificación podría ser dividir a los participantes en dos grupos: uno de verificadores experientes y otro sin experiencia.

Cada ejecución del experimento que se realiza en una unidad experimental es llamada **experimento unitario** o experimento elemental. Lo que significa que cada aplicación de una combinación de alternativas de factores por un sujeto experimental en una unidad experimental es un experimento elemental.

Un experimento elemental es cada terna $<analgésico_i, enfermero_j, paciente_k>$

para el ejemplo de *Efec-Analgésicos*. Para el ejemplo de *Efec-Técnicas* sería la terna $<técnica_i, verificador_j, software_k>$.

La figura 2.1 ilustra la interacción entre los distintos tipos de componentes de un experimento.

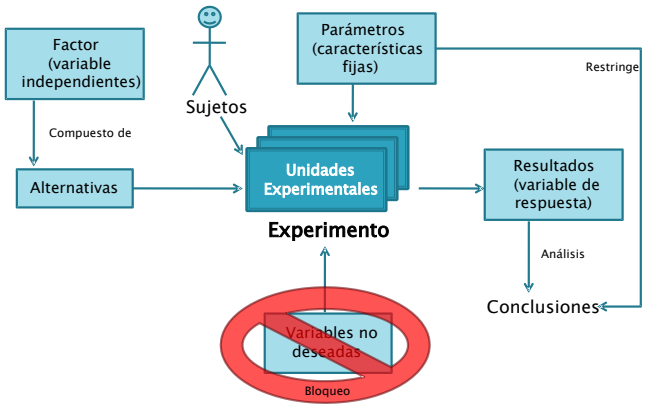



Figura 2.1: Componentes en un experimento de Ingeniería de Software

### 2.3.2. Principios generales de diseño

Muchos aspectos deben ser tenidos en cuenta cuando se diseña un experimento. Los principios generales de diseño son: aleatoriedad, bloqueo y balance. A continuación se describe en qué consiste cada principio.

**Aleatoriedad:** el principio de aleatoriedad es uno de los principios de diseño más importantes. Todos los métodos de análisis estadístico requieren que las observaciones sean de variables independientes aleatorias. Por consiguiente, tanto las alternativas de los factores como los sujetos tienen que ser elegidos de forma aleatoria, ya que los sujetos tienen un impacto crítico en el valor de las variables de respuesta.

La aleatoriedad que se puede aplicar a un experimento también depende del tipo de diseño que se haya elegido. Por ejemplo, si se tienen dos factores A y B, cada uno con dos posibles alternativas (a1, a2, b1 y b2), las alternativas deben ser combinadas de la siguiente forma: a1b1, a1b2, a2b1, a2b2, ya que cuando se tienen dos factores se quiere observar el efecto de cada alternativa por separado y de la interacción entre ambas.

Esta combinación de alternativas es especificada por el tipo de diseño experimental que se eligió. Sin embargo, las cuatro combinaciones deben ser asignadas de forma aleatoria a los proyectos y sujetos, y es ahí en donde la aleatoriedad se aplica.

**Bloqueo:** la técnica de bloqueo se usa cuando se tienen factores que probablemente tengan efectos indeseados en las variables de respuesta y éstos efectos son conocidos y controlables.

Como se mencionaba en el ejemplo de *Efec-Técnicas* en la sección anterior, algunos verificadores podrían tener experiencia en el uso de las técnicas de verificación y otros no. Entonces, para minimizar el efecto de la experiencia, se agrupan a los participantes en dos grupos, uno con verificadores experientes y otro sin experiencia.

**Balance:** el balance es deseable ya que simplifica y fortalece el análisis estadístico de los datos, aunque no es necesario. Tomando como ejemplo el experimento de *Efec-Analgésicos* nuevamente, sería deseable que la cantidad de personas a las cuales se les administra Zolben sea igual a la cantidad de personas que se les administra Aspirina.

### 2.3.3. Tipos de Diseño

En el proceso del diseño experimental, primero se debe decidir (basándose en los objetivos del experimento) a qué factores y alternativas estarán sujetas las unidades experimentales y qué parámetros deben ser establecidos. Luego, se debe examinar si existe la posibilidad de que algunos de los parámetros no pueda mantenerse en un valor constante, en ese caso se debe tener en cuenta cualquier variación indeseable. Finalmente, se debe elegir qué variables de respuesta serán medidas y cuáles serán los objetos y sujetos experimentales.

Teniendo establecidos los parámetros, factores, variables de bloqueo y variables de respuesta, se debe elegir el tipo de diseño experimental, en el cual se establece cuántas combinaciones de experimentos unitarios y alternativas deben haber.

Los distintos tipos de diseño experimental dependen del objetivo del experimento, del número de factores, de las alternativas de los factores y de la cantidad de variaciones indeseadas, entre otros.

Los tipos de diseño experimental se dividen en diseños de *un solo factor* y diseños de *múltiples factores*. A continuación se profundiza en los experimentos de un solo factor.

#### Diseño de un solo factor (*One-Factor Design*)

Para experimentos con un solo factor existen distintos tipos de diseños estándar, los principales son: los completamente aleatorios y los aleatorios con comparación por pares.

Los diseños **completamente aleatorios** son los tipos de diseño más simples, en los cuales se intenta comparar dos o más alternativas aplicadas a un determinado número de unidades experimentales, en donde cada unidad experimental se ve afectada una única vez, y por ende, por una sola alternativa. La asignación de las alternativas a los experimentos debe ser de forma aleatoria para asegurar la validez del análisis de datos.

Tomando como ejemplo *Efec-Técnicas* y suponiendo que el conjunto de programas sobre el cual se quiere conocer la efectividad de las técnicas lo componen diez programas distintos, se tendría que asignar las técnicas y los ingenieros de forma aleatoria a los programas que se vayan a verificar.

Una posible asignación aleatoria sería tener en una bolsa los nombres de todas las técnicas de verificación a aplicar, en donde la primera que se extraiga se aplique al programa $P_1$, la segunda a $P_2$ y así hasta el programa $P_{10}$. Luego de tener las duplas Programa-Técnica, efectuar la misma asignación aleatoria con los participantes: el primer participante extraído se lo asigna la dupla $(P_1, T_x)$, el segundo a la dupla $(P_2, T_y)$, y así sucesivamente.

El análisis estadístico que se puede hacer a este tipo de experimentos varía según si se aplican 2 o más alternativas para el factor.

Los diseños **aleatorios con comparación por pares** tienen como objetivo encontrar cuál es la mejor alternativa respecto de una determinada variable de respuesta. Estos tipos de diseño tienen la particularidad de que las alternativas se aplican al mismo experimento, instanciado en más de una unidad experimental.

Para el experimento de *Efec-Técnicas* no sería una buena decisión que cada ingeniero verificara 2 veces el mismo programa. En la segunda instancia de verificación, el ingeniero posee conocimiento tanto de los defectos del programa como de la tarea de verificar propiamente dicha (aunque sea con una técnica distinta). Por esto, para comparar las dos técnicas, ambas tienen que ser aplicadas por primera vez por ingenieros distintos, pero con similares características (ya que encontrar uno igual es imposible). La alternativa que debe aplicar cada ingeniero al programa debe ser asignada de forma aleatoria y no debe verificar un mismo programa más de una vez.

En este tipo de diseños se bloquean cierto tipo de variables que representan restricciones en la aleatoriedad que se le puede dar. Tomando como ejemplo nuevamente a *Efec-Técnicas*, si un verificador sin experiencia aplica más de una técnica durante el experimento, no sería deseable asignar al azar la técnica que cada verificador aplica en cada verificación.

Existe un efecto de aprendizaje en el cual, luego de que un verificador ejecutó una verificación, éste generó conocimiento sobre la verificación en sí, independientemente de la técnica que haya aplicado, y éste conocimiento influye significativamente en la segunda instancia de verificación que vaya a aplicar. Por tanto, la aleatoriedad en el orden de la asignación de técnicas en este ejemplo no es del todo deseable.

## 2.4. Proceso Experimental

Como se mencionó anteriormente, los experimentos son una técnica de investigación en la cual se quiere tener un mejor control del estudio y del entorno en el que éste se lleva a cabo.

Los experimentos son apropiados para investigar distintos aspectos de la IS, como ser: confirmar teorías, explorar relaciones, evaluar la exactitud de los modelos y validar medidas. Tienen un alto costo respecto de las otras técnicas de investigación, pero a cambio ofrecen un control total de la ejecución y son de fácil replicación.

El proceso para llevar a cabo un experimento está formado por varias fases: definición, planificación, operación, análisis e interpretación y presentación.

La primer fase es la de **definición**, en donde se define el experimento en términos del problema, objetivos y metas. La siguiente fase es la **planificación**, en la cual se determina el diseño del experimento. En la fase de **operación** se ejecuta el diseño del experimento, en donde se recolectan los datos que serán analizados posteriormente en la fase de **análisis e interpretación**. En esta última fase, conceptos estadísticos son aplicados para analizar los datos. Por último, se muestran los resultados obtenidos en la fase de **presentación**.

En la figura 2.2 se muestra una visión general de todo el proceso. Cada una de las fases que lo componen se detallan a continuación.

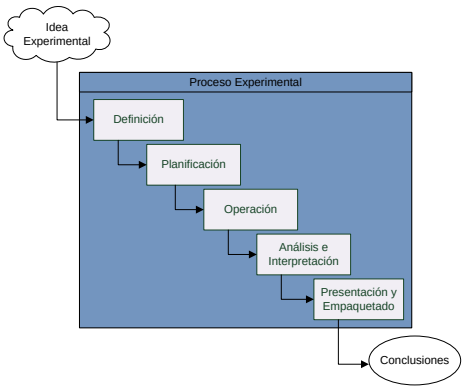



Figura 2.2: Visión general del Proceso Experimental

### 2.4.1. Definición

En la fase de Definición se determinan las bases del experimento, que se ilustra en la figura 2.3. Para ello se debe definir **el problema que se quiere resolver**, **propósito del experimento** y **los objetivos y metas** del mismo.

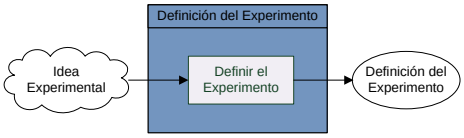



Figura 2.3: Fase de Definición del Experimento

Para el planteo del objetivo del experimento se debe definir *el objeto de estudio*, que es la entidad que va a ser estudiada en el experimento. Puede ser un producto, proceso, recurso u otro. También se debe establecer el *propósito*: la intención del experimento. Por ejemplo, evaluar diferentes técnicas de verificación.

Se debe definir además el *foco de calidad*, que refiere al efecto primario que esta bajo estudio, ejemplos son la efectividad y el costo de las técnicas de verificación. El propósito y el foco de calidad son las bases para las hipótesis del experimento.

Otro aspecto que debe estar presente es la *perspectiva*, que refiere al punto de vista con que los resultados obtenidos son interpretados. Por ejemplo, los resultados de la comparación de técnicas de verificación pueden verse desde la perspectiva de un experimentador, de un investigador o de un profesional. Un experimentador verá el estudio como una demostración de como una técnica de verificación puede ser evaluada. Un investigador puede ver el estudio como una base empírica para refinar teorías sobre la verificación de software, enfocándose en los datos que apoyan o refutan estas teorías. Un profesional puede ver el estudio como una fuente de información sobre qué técnicas de verificación deberían aplicarse en la práctica.

Junto con los aspectos mencionados se debe definir el *contexto*, que es el ambiente en el que se ejecuta el experimento. En este punto se deben definir los *sujetos* que van a llevar a cabo el experimento y los *artefactos* que son utilizados

en la ejecución del mismo. Se puede caracterizar el contexto de un experimento según el número de sujetos y objetos definidos en él: un solo objeto y un solo sujeto, un solo sujeto a través de muchos objetos, un solo objeto a través de un conjunto de sujetos, o un conjunto de sujetos y un conjunto de objetos.

### 2.4.2. Planificación

La planificación es la fase en la que se define como se va a llevar a cabo el experimento. Esta fase consta de las etapas: selección del contexto, formulación de las hipótesis, elección de las variables, selección de los sujetos, diseño del experimento, instrumentación y evaluación de la validez, que se muestran en la figura 2.4.

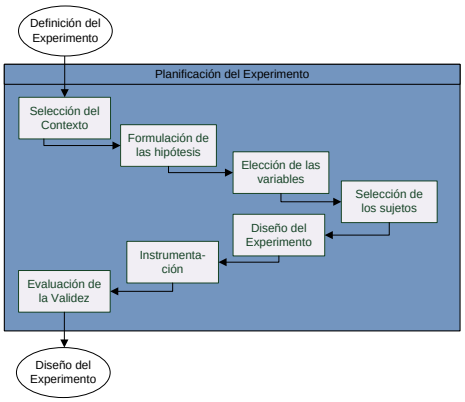



Figura 2.4: Fase de Planificación del Experimento

La etapa de **selección del contexto** es la etapa inicial de la planificación. En esta etapa se amplía el contexto definido en la etapa de Definición, especificando claramente las características del ambiente donde ejecuta el experimento. Se define si el experimento se va a realizar en un proyecto real (en línea, *on-line*) o en un laboratorio (fuera de línea, *off-line*), características de los sujetos y si el problema es "real" (problema existente en la industria) o "de juguete". También se debe definir si el experimento es válido para un contexto específico o para un dominio general de la Ingeniería de Software.

Una vez que los objetivos están claramente definidos se pueden transformar en una hipótesis formal. La **formulación de las hipótesis** es una fase muy importante dentro de la etapa de planificación, ya que la verificación de la misma es la base para el análisis estadístico. En esta fase se formaliza la definición del experimento en la hipótesis.

Usualmente se definen dos hipótesis, la hipótesis nula y la hipótesis alternativa. La hipótesis nula, denotada $H_0$, asume que no hay una diferencia significativa entre las alternativas, con respecto a las variables dependientes que se están midiendo. Establece que si hay diferencias entre las observaciones realizadas, éstas son por casualidad, no producto de la alternativa aplicada. Esta hipótesis se asume verdadera hasta que los datos demuestren lo contrario, por lo que el foco del experimento está puesto en rechazarla. Un ejemplo de hipótesis nula es: "No hay diferencia en la cantidad de defectos encontrados por las técnicas de verificación".

En cambio la hipótesis alternativa, denotada $H_1$, afirma que existe una diferencia significativa entre las alternativas con respecto a las variables dependientes. Establece que las diferencias encontradas son producto de la aplicación de las alternativas. Ésta es la hipótesis a probar, para esto se debe determinar que los datos obtenidos son lo suficientemente convincentes para desechar la hipótesis nula y aceptar la hipótesis alternativa. Un ejemplo de hipótesis alternativa es, si se están comparando dos técnicas de verificación, decir que una encuentra más defectos que la otra. En caso de haber más de una hipótesis alternativa se denotan secuencialmente: $H_1$, $H_2$, $H_3$, ..., $H_n$.

Una vez definida la hipótesis, se debe identificar qué variables afectan a la/s alternativa/s. Luego de identificadas las variables se debe decidir el control a ejercer sobre las mismas.

La **selección de las variables** dependientes como la de las independientes están relacionadas, por lo que en muchos casos se realizan en simultáneo. Seleccionar estas variables es una tarea muy compleja, que en ocasiones implica conocer muy bien el dominio. Es importante definir las variables independientes y analizar sus características, para así investigar y controlar los efectos que ejercen sobre las variables dependientes. Se deben identificar las variables independientes que se pueden controlar y las que no. Además, se deben identificar las variables dependientes, mediante las cuales se mide el efecto de las alternativas. Generalmente hay sólo una variable dependiente y se deriva de la hipótesis.

Otro aspecto importante al llevar a cabo un experimento es la **selección de los sujetos**. Para poder generalizar los resultados al resto de la población, la selección debe ser una muestra representativa de la misma. Cuanto más grande es la muestra, menor es el error al generalizar los datos. Existen dos tipos de muestras que se pueden seleccionar: la probabilística, donde se conoce la probabilidad de seleccionar cada sujeto; y la no-probabilística, donde esta probabilidad es desconocida.

Luego de definir el contexto, formalizar las hipótesis, y seleccionar las variables y los sujetos, se debe **diseñar el experimento**. Es muy importante planear y diseñar cuidadosamente el experimento, para que los datos obtenidos puedan ser interpretados mediante la aplicación de métodos de análisis estadísticos.

Para comenzar a diseñar un experimento se debe elegir el diseño adecuado. Se debe planificar y diseñar el conjunto de las combinaciones de alternativas, sujetos y objetos, que conforman los experimentos unitarios. Se describe cómo estos experimentos unitarios deben ser organizados y ejecutados.

La elección del diseño del experimento afecta el análisis estadístico y viceversa, por lo que al elegir el diseño del experimento se debe tener en cuenta qué análisis estadístico es el mejor para rechazar la hipótesis nula y aceptar la alternativa.

Luego de diseñar el experimento y antes de la ejecución es necesario contar con todo lo necesario para la correcta ejecución del mismo. La **instrumentación** involucra, de ser necesario, capacitación a los sujetos, preparación de los artefactos, construcción de guías, descripción de procesos, planillas y herramientas. También implica configurar el hardware, mecanismos de consultas y experiencias piloto, entre otros. La finalidad de esta fase es proveer todo lo necesario para la realización y monitorización del experimento.

### 2.4.3. Evaluación de la Validez

Una pregunta fundamental antes de pasar a ejecutar el experimento es cuán válidos serían los resultados. Existen cuatro categorías de amenazas a la validez: validez de la conclusión, validez interna, validez del constructo y validez externa.

Las amenazas que afectan la **validez de la conclusión** refieren a las conclusiones estadísticas. Amenazas que afecten la capacidad de determinar si existe una relación entre la alternativa y el resultado, y si las conclusiones obtenidas al respecto son válidas. Ejemplos de estas son la elección de los métodos estadísticos, y la elección del tamaño de la muestra, entre otros.

Las amenazas que influyen en la **validez interna** son aquellas referidas a observar relaciones entre la alternativa y el resultado que sean producto de la casualidad y no del resultado de la aplicación de un factor. Esta "casualidad" es provocada por elementos desconocidos que influyen sobre los resultados sin el conocimiento de los investigadores. Es decir, la validez interna se basa en asegurar que la alternativa en cuestión produce los resultados observados.

La **validez del constructo** indica cómo una medición se relaciona con otras de acuerdo con la teoría o hipótesis que concierne a los conceptos que se están midiendo. Un ejemplo se puede observar al momento de seleccionar los sujetos en un experimento. Si se utiliza como medida de la experiencia del sujeto el número de cursos que tiene aprobados en la universidad, no se está utilizando una buena medida de la experiencia. En cambio, una buena medida puede ser utilizar la cantidad de años de experiencia en la industria o una combinación de ambas cosas.

La **validez externa** está relacionada con la habilidad para generalizar los resultados. Se ve afectada por el diseño del experimento. Los tres riesgos principales que tiene la validez externa son: tener los participantes equivocados como sujetos, ejecutar el experimento en un ambiente erróneo y realizar el experimento en un momento que afecte los resultados.

### 2.4.4. Operación

Luego de diseñar y planificar el experimento, éste debe ser ejecutado para recolectar los datos que se quieren analizar. La operación del experimento consiste en tres etapas: preparación, ejecución y la validación de los datos, que se muestran en la figura 2.5.

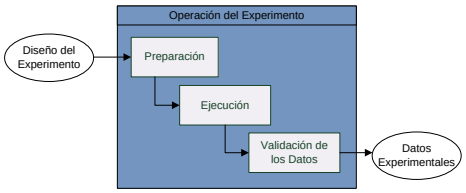



Figura 2.5: Fase de Operación del Experimento

En la etapa de preparación se seleccionan los sujetos y se preparan los artefactos a ser utilizados.

Es muy importante que los sujetos estén motivados y dispuestos a realizar las actividades que les sean asignadas, ya sea que tengan conocimiento o no de

su participación en el experimento. Se debe intentar obtener consentimiento por parte de los participantes, que deben estar de acuerdo con los objetivos de la investigación. Los resultados obtenidos pueden volverse inválidos si los sujetos al momento que deciden participar no saben lo que tienen que hacer o tienen un concepto erróneo.

Es importante considerar la sensibilidad de los resultados que se obtienen de los sujetos, por ejemplo: es importante asegurar a los participantes que los resultados obtenidos sobre su rendimiento se mantienen en secreto y no se usarán para perjudicarlos en ningún sentido. Se debe tener en cuenta también los incentivos, ya que ayudan a motivar a los sujetos, pero se corre el riesgo de que participen sólo por el incentivo, lo que puede ser perjudicial para el experimento. En caso de no tener otra alternativa que no sea engañar a los sujetos, se debe procurar explicar y revelar el engaño lo más temprano posible.

Como se vio en la instrumentación, para que los sujetos comiencen la ejecución es necesario tener prontos todos los instrumentos, formularios, herramientas, guías y otros artefactos que sean necesarios para la ejecución del experimento. Muchas veces se debe preparar un conjunto de instrumentos especial para cada sujeto y otras se utiliza el mismo conjunto de artefactos para todos los sujetos.

Existen muchas formas distintas de ejecutar los experimentos, la duración varía desde días hasta años.

Los datos pueden ser recolectados de las siguientes formas:

- Manualmente mediante el llenado de formularios por parte de los sujetos.
- Manualmente soportado por herramientas.
- Mediante entrevistas.
- Automáticamente por herramientas.

La primera es la forma más común y no requiere mucho esfuerzo por parte del experimentador. Tanto en los formularios como en los métodos soportados por herramientas no es posible identificar inconsistencias o defectos hasta que no se recolecte la información, o hasta que los sujetos los descubran. En las entrevistas, el contacto con los sujetos es mucho mayor permitiendo una mejor comunicación con ellos durante la recolección de datos. Éste método es el que requiere mas esfuerzo por parte del investigador.

Un aspecto muy importante a la hora de ejecutar los experimentos es el ambiente de ejecución, tanto si el experimento se realiza dentro de un proyecto de desarrollo común o si se crea un ambiente ficticio para su ejecución. En el primer caso el experimento no debería afectar el proyecto más de lo necesario, ya que la razón de realizar el experimento dentro de un proyecto es ver los efectos de las alternativas en el ambiente del proyecto. Si el experimento cambia demasiado el ambiente del proyecto, éstos efectos se perderían.

Cuando se obtienen los datos, se debe chequear que fueron recolectados correctamente y que son razonables. Algunas fuentes de error son que los sujetos llenen mal sus planillas, o no recolecten los datos seriamente, lo que hace que se descarten datos. Es importante revisar que los sujetos hagan un trabajo serio y responsable y que apliquen las alternativas en el orden correcto, en otras palabras: que el experimento sea ejecutado en la forma en que fue planificado. De lo contrario los resultados podrían ser inválidos.

### 2.4.5. Análisis e Interpretación

Luego de que finaliza la ejecución del experimento y se cuenta con los datos recolectados, comienza la fase de análisis de los mismos conforme a los objetivos planteados.

Un aspecto importante a considerar en el análisis de los datos es la **escala de medida**. La escala de medida utilizada para recolectar los datos restringe el tipo de cálculos estadísticos que se pueden realizar. Una medida es un mapeo de un atributo de una entidad a un valor de medida, por lo general un valor numérico. Las entidades son objetos que se observan en la realidad, por ejemplo, una técnica de verificación.

El propósito de mapear los atributos en un valor de medida es caracterizar y manipular los atributos formalmente. La medida seleccionada debe ser válida, por tanto, no debe violar ninguna propiedad necesaria del atributo que mide, y debe ser una caracterización matemática adecuada del atributo.

El mapeo de un atributo a un valor de medida puede realizarse de varias formas. Cada tipo de mapeo posible de un atributo se conoce como escala. Los tipos más comunes de escala son:

- Escala Nominal.- Es la menos poderosa de las escalas. Solo mapea el atributo de la entidad en un nombre o símbolo. El mapeo puede verse como una clasificación de las entidades acorde al atributo. Ejemplos de escala nominal son clasificaciones, etiquetados, etc.

- Escala Ordinal.- La escala ordinal categoriza las entidades según un criterio de ordenación. Es más poderosa que la escala nominal. Ejemplos de criterios de ordenación son “mayor que”, “mejor que” y “más complejo”. Ejemplos de escala nominal son grados, complejidad del software, entre otras.

- Escala de intervalo.- La escala de intervalo se utiliza cuando la diferencia entre dos medidas es significativa, pero no el valor en si mismo. Este tipo de escala ordena los valores de la misma forma que la escala ordinal, pero existe la noción de “distancia relativa” entre dos entidades. Esta escala es más poderosa que la ordinal. Ejemplos de escala de intervalo son la temperatura medida en Celsius o Fahrenheit.

- Escala ratio (cociente de dos números).- Si existe un valor cero significativo y la división entre dos medidas es significativa, se puede utilizar una escala ratio. Ejemplos de escala ratio son distancia, temperatura medida en Kelvin, etc.

Después de obtener los datos es necesario interpretarlos para llegar a conclusiones válidas. La interpretación se realiza en tres etapas: caracterizar el conjunto de datos usando estadística descriptiva, reducción del conjunto de datos y realización de las pruebas de hipótesis que se ilustran en la figura 2.6.

#### Estadística Descriptiva

La **estadística descriptiva** se utiliza antes de la prueba de hipótesis, para entender mejor la naturaleza de los datos y para identificar datos falsos o anormales. Los aspectos principales que se examinan son: la tendencia central,

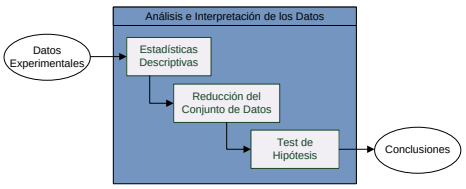



Figura 2.6: Fase de Análisis e Interpretación de los Datos del Experimento

la dispersión y la dependencia. A continuación se presentan las medidas más comunes de cada uno de estos aspectos. Para ello se asume que existen $x_1 \dots x_n$ muestras.

Las **medidas de tendencia central** indican "el medio" de un conjunto de datos. Entre las medidas más comunes se encuentran: la media aritmética, la mediana y la moda.

La *media aritmética* se conoce como el promedio, y se calcula sumando todas las muestras y dividiendo el total por el número de muestras:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.1}$$

La *media*, denotada $\bar{x}$, resume en un valor las características de una variable teniendo en cuenta a todos los casos. Es significativa para las escalas de intervalo y ratio.

La *mediana*, denotada $\tilde{x}$, representa el valor medio de un conjunto de datos, tal que el número de muestras que son mayores que la mediana es el mismo que el número de muestras que son menores que la mediana. Se calcula ordenando las muestras en orden ascendente o descendente, y seleccionando la observación del medio. Este cálculo está bien definido si $n$ es impar. Si $n$ es par, la mediana se define como la media aritmética de los dos valores medios. Esta medida es significativa para las escalas ordinal, de intervalo y ratio.

La *moda* representa la muestra más común. Se calcula contando el número de muestras para cada valor único y seleccionando el valor con más cantidad. La moda esta bien definida si hay solo un valor más común que los otros. Si este no es el caso, se calcula como la mediana de las muestras más comunes. La moda es significativa para las escalas nominal, ordinal, de intervalo y ratio.

La media aritmética y la mediana son iguales si la distribución de las muestras es simétrica. Si la distribución es simétrica y tiene un único valor máximo, las tres medidas son iguales.

Las medidas de tendencia central no proveen información sobre la dispersión del conjunto de datos. Cuanto mayor es la dispersión, más variables son las muestras, cuanto menor es la dispersión, más homogéneas a la media son las muestras.

Las **medidas de dispersión** miden el nivel de desviación de la tendencia central, o sea, que tan diseminados o concentrados están los datos respecto al valor central. Entre las principales medidas de dispersión están: la varianza, la desviación estándar, el rango y el coeficiente de variación.

La *varianza* ($s^2$) que presenta una distribución respecto de su media se calcula como la media de las desviaciones de las muestras respecto a la media

aritmética. Dado que la suma de las desviaciones es siempre cero, se toman las desviaciones al cuadrado:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{2.2}$$

Se divide por $n-1$ y no por $n$, porque dividir por $n-1$ provee a la varianza de propiedades convenientes. La varianza es significativa para las escalas de intervalo y ratio.

La *desviación estándar*, denotada $s$, se de
ne como la raíz cuadrada de la varianza:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2.3}$$

A menudo esta medida se prefiere sobre la varianza porque tiene las mismas dimensiones (unidad de medida) que los valores de las muestras. En cambio, la varianza se mide en unidades cuadráticas. La desviación estandar es significativa para las escalas de intervalo y ratio.

La dispersión también se puede expresar como un porcentaje de la media. Este valor se llama *coeficiente de variación*, y se calcula como:

$$100 \cdot \frac{s}{\bar{x}} \tag{2.4}$$

Esta medida no tiene dimensión y es significativa para la escala ratio. Permite comparar la dispersión o variabilidad de dos o más grupos.

El **rango** de un conjunto de datos es la distancia entre el valor máximo y el mínimo:

$$range = x_{max} - x_{min} \tag{2.5}$$

Es una medida significativa para las escalas de intervalo y ratio. Cuando el conjunto de datos consiste en muestras relacionadas de a pares (xi; yi) de dos variables, X e Y, puede ser interesante examinar la dependencia entre estas variables. Las principales medidas de dependencia son: regresión lineal, covarianza y el coefiente de correlación lineal.

### Reducción del Conjunto de Datos

Para las pruebas de hipótesis se utilizan métodos estadísticos. El resultado de aplicar estos métodos depende de la calidad de los datos. Si los datos no representan lo que se cree, las conclusiones que se derivan de los resultados de los métodos son incorrectas. Errores en el conjunto de datos pueden ocurrir por un error sistemático, o por lo que se conoce en estadística con el nombre de outlier. Un outlier es un dato mucho más grande o mucho más chico de lo que se puede esperar observando el resto de los datos.

Las estadísticas descriptivas se ven fuertemente influenciadas por aquellas observaciones que su valor dista significativamente del resto de los valores recolectados. Estas observaciones llevan el nombre de *outliers*.

Los *outliers* influencian las medidas de dispersión, aumentando la variabilidad de lo que se está midiendo. En algunos casos se realiza un análisis acerca de

estos valores que difieren mucho de la media y se decide quitarlos de los datos a analizar porque no son representativos de la población, ya que fueron causados por algún tipo de anomalía: errores de medición, variaciones no deseadas en las características de los sujetos, entre otras.

Quitar *outliers* requiere de un análisis pormenorizado, por quitar outliers se demoró en detectar el agujero de la capa de ozono. [2]

Una vez identificado un outlier se debe identificar su origen para decidir qué hacer con él. Si se debe a un evento raro o extraño que no volverá a ocurrir, el punto puede ser excluido. Si se debe a un evento extraño que puede volver a ocurrir, no es aconsejable excluir el valor del análisis, pues tiene información relevante. Si se debe a una variable que no fue considerada, debería ser considerado para basar los cálculos y modelos también en esta variable.

### Pruebas de Hipótesis

El objetivo de la **prueba de hipótesis** es ver si es posible rechazar cierta hipótesis nula $H_0$. Si la hipótesis nula no es rechazada, no se puede decir nada sobre los resultados. En cambio, si es rechazada, se puede declarar que la hipótesis es falsa con una significancia dada ($\alpha$). Este nivel de significancia también es denominado nivel de riesgo o probabilidad de error, ya que se corre el riesgo de rechazar la hipótesis nula cuando en realidad es verdadera. Este nivel está bajo el control del experimentador.

Para probar $H_0$ se define una unidad de prueba $t$ y un área crítica $C$, la cual es parte del área sobre la que varía $t$. A partir de estas definiciones se formula la prueba de significancia de la siguiente forma:

- Si $t \in C$, rechazar $H_0$
- Si $t \notin C$, no rechazar $H_0$

Por ejemplo, un experimentador observa la cantidad de defectos detectados por LOC de una técnica de verificación desconocida bajo determinadas condiciones, y quiere probar que no es la técnica B, de la cual sabe que en las mismas condiciones (programa, verificador, etc.) detecta 1 defecto cada 20 LOC. El experimentador sabe que también pueden haber otras técnicas que detecten 1 defecto cada 20 LOC. A partir de esto se define la hipótesis nula: "$H_0$: La técnica observada es la B". En este ejemplo, la unidad de prueba $t$ es cada cuantos LOC se detecta un defecto y el área crítica es $C = \{1, 2, 3, \ldots, 19, 21, 22, \ldots\}$. La prueba de significancia es: si $t \leq 19$ o $t \geq 21$, rechazar $H_0$, de lo contrario no rechazar $H_0$.

Si se observa que $t = 20$, la hipótesis no puede ser rechazada ni se pueden derivar conclusiones, pues pueden haber otras técnicas que detecten un defecto cada 20 LOC.

El área crítica, $C$, puede tener distintas formas, lo más común es que tenga forma de intervalo, por ejemplo: $t \leq a$ o $t \geq b$. Si $C$ consiste en uno de estos intervalos es unilateral. Si consiste de dos intervalos ($t \leq a$, $t \geq b$, donde $a < b$), es bilateral.

[2] En 1985 los científicos británicos anunciaron un agujero en la capa de ozono sobre el polo sur. El reporte fue descartado ya que observaciones más completas, obtenidas por instrumentos satelitales, no mostraban nada inusual. Luego, un análisis más exhaustivo reveló que las lecturas de ozono en el polo sur eran tan bajas que el programa que las analizaba las había suprimido automáticamente como outliers en forma equivocada.

Hay varios métodos estadísticos, de aquí en adelante denotados *tests*, que pueden utilizarse para evaluar los resultados de un experimento, más específicamente para determinar si se rechaza la hipótesis nula. Cuando se lleva a cabo un *test* es posible calcular el menor valor de significancia posible (denotado *p*-valor) con el cual es posible rechazar la hipótesis nula. Se rechaza la hipótesis nula si el p-valor asociado al resultado observado es menor o igual que el nivel de significancia establecido.

Las siguientes son tres probabilidades importantes para la prueba de hipótesis:

- $\alpha = P(\text{cometer el error tipo I}) = P(\text{rechazar } H_0 | H_0 \text{ es verdadera})$. Es la probabilidad de rechazar $H_0$ cuando es verdadera.
- $\beta = P(\text{cometer el error tipo II}) = P(\text{no rechazar } H_0 | H_0 \text{ es falsa})$. Es la probabilidad de no rechazar $H_0$ cuando es falsa.
- $\text{Poder} = 1 - \beta = P(\text{rechazar } H_0 | H_0 \text{ es falsa})$. El poder de prueba es la probabilidad de rechazar $H_0$ cuando es falsa.

El experimentador debería elegir un test con un poder de prueba tan alto como sea posible. Hay varios factores que afectan el poder de un test. Primero, el test en sí mismo puede ser más o menos efectivo. Segundo, la cantidad de muestras: mayor cantidad de muestras equivale a un poder de prueba más alto. Otro aspecto es la selección de una hipótesis alternativa unilaterial o bilateral. Una hipótesis unilateral da un poder mayor que una bilateral.

La probabilidad de cometer un error tipo I se puede controlar y reducir. Si la probabilidad es muy pequeña, sólo se rechazará la hipótesis nula si se obtiene evidencia muy contundente en contra de esta hipótesis. La probabilidad máxima de cometer un error tipo I se conoce como la significancia de la prueba ($\alpha$).

Los valores de uso más común para la significancia de una prueba son 0.01, 0.05 y 0.10. La significancia es en ocasiones presentada como un porcentaje, tal como 1 %, 5 % o 10 %. Esto quiere decir que el experimentador está dispuesto a permitir una probabilidad de 0.01, 0.05, o 0.10 de rechazar la hipótesis nula cuando es cierta, o sea, de cometer un error tipo I.

El valor de la significancia es seleccionado antes de comenzar a hacer el experimento en una de varias formas.

El valor de $\alpha$ puede estar establecido en el área de investigación, por ejemplo: se puede obtener de artículos que se publican en revistas científicas. Otra forma de seleccionarlo es que sencillamente sea impuesto por la persona o compañía para la cual se trabaja. Finalmente, puede ser seleccionado tomando en cuenta el costo de cometer un error tipo I. Mientras más alto el costo, más pequeña debe ser la probabilidad $\alpha$ de cometer un error tipo I. El valor usual de $\alpha$ en las ciencias naturales y sociales es de 0.05. En Ingeniería de Software, el valor de $\alpha$ aún no se encuentra establecido.

Existen dos tipos de tests: paramétricos y no paramétricos. Los **tests paramétricos** están basados en un modelo que involucra una distribución específica. En la mayoría de los casos, se asume que algunos de los parámetros involucrados en un test paramétrico están normalmente distribuidos. Los tests paramétricos también requieren que los parámetros puedan ser medidos al menos en una *escala de intervalo*. Si los parámetros no pueden medirse en al menos una escala de intervalo, generalmente no se puede utilizar un test paramétrico. En este caso hay un amplio rango de tests no paramétricos disponible.

Los **tests no paramétricos** no asumen lo mismo respecto a la distribución de los parámetros, son más generales que los paramétricos. Un test no paramétrico se puede utilizar en vez de un test paramétrico, pero el caso inverso no siempre puede darse.

En la elección entre un test paramétrico y un test no paramétrico hay dos aspectos a considerar:

- Aplicabilidad.- Es importante que las suposiciones en cuanto a las distribuciones de parámetros y las que conciernen a las escalas sean realistas.
- Poder.- El poder de los tests paramétricos es generalmente mayor que el de los tests no paramétricos. Por lo tanto, los test paramétricos requieren menos datos (experimentos más pequeños), que los tests no paramétricos, siempre que sean aplicables.

Aunque es un riesgo utilizar tests paramétricos cuando no se cuenta con las condiciones requeridas, en algunos casos vale la pena tomar el riesgo. Algunas simulaciones han mostrado que los tests paramétricos son bastante robustos a las desviaciones de las pre-condiciones (escala de intervalo), mientras las desviaciones no sean demasiado grandes.

En el caso de las pruebas paramétricas, se exige que la distribución de la muestra se aproxime a una normal. Para poder utilizar aproximación normal se requiere un tamaño mínimo de la muestra, dependiendo del $p(value)$ que se requiera [17]. En el cuadro 2.1 se muestran los tamaños mínimos de muestra para los distintos $p(value)$.

| **p(value)** | **Tamaño mínimo de muestra** |
|---|---|
| 0.50 | n = 30 |
| 0.40 ó 0.60 | n = 50 |
| 0.30 ó 0.70 | n = 80 |
| 0.20 ó 0.80 | n = 200 |
| 0.10 - 0.90 | n = 600 |

Cuadro 2.1: Estadísticas descriptivas de la Efectividad

Los test paramétricos más usados en experimentos de Ingeniería de Software son:

- ANOVA (*ANalysis Of VAriance*) [24].
- ANOM (*ANalysis Of Means*) [16].

Ambos tests (ANOVA y ANOM), pueden utilizarse para diseños de un solo factor con múltiples alternativas. En ambos test la hipótesis nula refiere a la igualdad de las medias (como es habitual en los test paramétricos):

$H_0 : \bar{x}_1 = \bar{x}_2 = \ldots = \bar{x}_I$

En ANOVA, la variación en la respuesta se divide en la variación entre los diferentes niveles del factor (los diferentes tratamientos) y la variación entre individuos dentro de cada nivel. El objetivo principal del ANOVA es contrastar si existen diferencias entre las diferentes medias de los niveles de las variables (factores).

En el caso de ANOM, este test no solamente responde a la pregunta de si hay o no diferencias entre las alternativas, sino que cuando hay diferencias, también dice cuáles alternativas son mejores y cuáles peores.

Los test no paramétricos más usado son:

- Kruskal Wallis.
- Mann-Whitney.

En el caso de los test no paramétricos, la hipótesis nula refiere a la igualdad de las medianas:

$H_0 : \tilde{x}_1 = \tilde{x}_2 = \ldots = \tilde{x}_I$

Rechazar $H_0$ significa que existe evidencia estadística como para afirmar de que hay diferencias entre las alternativas. En el caso de que hubiera más de dos alternativas, para conocer cuál es la alternativa que difiere es necesario comparar las alternativas de a dos.

Si se presume que una alternativa puede ser mejor o peor que el resto, esto quiere decir que hay un "ordenamiento" entre ellas, lo aconsejable es realizar un test de ordenamiento. Algunos test de ordenamientos son:

- Jonckheere-Terpstra Test. [13]
- Test para alternativas ordenadas L. [13]

Para las pruebas de ordenamiento, las hipótesis que se plantean son las siguientes:

$H_0 : \tilde{x}_1 = \tilde{x}_2 = \ldots = \tilde{x}_I$

$H_1 : \tilde{x}_1 \leq \tilde{x}_2 \leq \ldots \leq \tilde{x}_I$ (con al menos una desigualdad estricta)

### 2.4.6. Presentación y Empaquetado

En la presentación y el empaquetado de un experimento es esencial no olvidar aspectos e información necesaria para que otros puedan replicar o tomar ventaja del experimento y del conocimiento ganado durante su ejecución.

El esquema de reporte de un experimento generalmente cuenta con los siguientes títulos: Introducción, Definición del Problema, Planificación del Experimento, Operación del Experimento, Análisis de Datos, Interpretación de Resultados, Discusión y Conclusiones, y Apéndice.

En la Introducción se realiza una introducción al área y los objetivos de la investigación. En la Definición del Problema se describe en mayor profundidad el trasfondo de la investigación, incluyendo las razones para realizarla. En Planificación del Experimento se detalla el contexto del experimento incluyendo las hipótesis, que se derivan de la definición del problema; las variables que se deben medir, tanto independientes como dependientes; la estrategia de medida y análisis de datos; los sujetos que participaran de la investigación; y las amenazas a la validez.

En la Operación del Experimento se describe como preparar la ejecución del mismo, incluyendo aspectos que permitan facilitar la replicación y descripciones que indiquen cómo se llevaron a cabo las actividades. Debe incluirse la preparación de los sujetos, cómo se recolectaron los datos y cómo se realizó la ejecución.

En el Análisis de Datos se describen los cálculos y los modelos de análisis específicos utilizados. Se debe incluir información, como por ejemplo, tamaño de la muestra, niveles de significancia y métodos estadísticos utilizados, para que el lector conozca los pre-requisitos para el análisis. En la Interpretación de los Resultados se rechaza la hipótesis nula o se concluye que no puede ser rechazada. Aquí se resume cómo utilizar los datos obtenidos en el experimento. La interpretación debe realizarse haciendo referencia a la validez. También se deben describir los factores que puedan tener un impacto sobre los resultados.

Finalmente, en Discusión y Conclusiones se presentan las conclusiones y los hallazgos como un resumen de todo el experimento, junto con los resultados, problemas, desviaciones respecto al plan, etc. También se incluyen ideas sobre trabajos a futuro. Los resultados deberían ser comparados con los obtenidos por trabajos anteriores, de manera de identificar similitudes y diferencias. La información que no es vital para la presentación se incluye en el Apéndice. Esto puede ser, por ejemplo, los datos recavados y más información sobre sujetos y objetos. Si la intención es generar un paquete de laboratorio, el material utilizado en el experimento puede ser proveído en el apéndice.

# Capítulo 3

# Marco de Comparación de Experimentos

Existen diversos experimentos formales para conocer cómo se comportan distintas técnicas de pruebas de software. Contar con un marco de comparación de experimentos es necesario para poder compararlos formalmente, y para poder avanzar en la construcción de nuevos experimentos basándose en experimentos anteriores. En este capítulo se presenta un marco de comparación y como ejemplo se utiliza el marco con algunos experimentos conocidos. Se detallan los experimentos elegidos para realizar el marco en orden cronológico, siendo el primero el realizado por Basili y Selby[2] en 1987, seguido por Kamsties y Lott [11] en 1995, Macdonald y Miller [12] en 1997, y por último los dos experimentos realizados por Juristo y Vegas [10] en 2001.

## 3.1. Experimento Basili - Selby (1987)

El objetivo del experimento de Basili es caracterizar la efectividad del testing en relación a varios agentes y las relaciones entre estos. Dichos agentes son: la técnica, el programa, el tipo de falta, y la experiencia de los sujetos. Específicamente el objetivo de este estudio comprende tres aspectos diferentes de pruebas de software:

- La eficacia en la detección de faltas.
- El costo de la detección de faltas.
- Los tipos de defectos detectados.

El primer objetivo se refiere a identificar cuáles son las técnicas que detectan la mayoría de las faltas de los programas. El segundo se refiere a conocer cuáles técnicas de verificación detectan las faltas con alta tasa (cantidad faltas/esfuerzo). La tercera se refiere a conocer si las técnicas tienden a capturar diferentes tipos de defectos.

El estudio realizado utiliza tres técnicas para la verificación del software: testing funcional, testing estructural y revisión de código. En el testing funcional se utiliza la especificación del programa para la construcción de casos de

prueba a través de métodos como partición de clases de equivalencia y análisis del valor límite. En el testing estructural un sujeto inspecciona el código fuente y a continuación elabora y ejecuta casos de prueba. Para estas pruebas se utiliza el criterio de cubrimiento de sentencias. En la revisión de código por etapas de abstracción un sujeto identifica los principales subprogramas en el software, determina sus funciones y compone una especificación con el fin de que el programa quede completamente determinado. Luego compara esta especificación con la especificación real del programa.

En este estudio se apunta a utilizar un entorno de prueba realista para evaluar la eficacia de las técnicas de verificación. Para esto, los sujetos son elegidos para ser representativos de los diferentes niveles de conocimiento. Los programas de prueba corresponden a diferentes tipos de software de programación común, y las faltas en los programas son representativas de las que frecuentemente se producen en el desarrollo de software.

El experimento consta de tres fases. La primera y segunda fase se realizan en la Universidad de Maryland en 1982 y 1983 respectivamente. La tercera fase se realiza en *Computer Sciences Corporation* (Silver Spring, MD) y la NASA **Goddard Space Flight Center** (Greenbelt, MD) en otoño de 1984. Se experimenta con un total de 74 sujetos, en cada una de las fases hay 29, 13, y 32 sujetos respectivamente. Estos fueron seleccionados para que representen los tres niveles diferentes de experiencia informática: avanzado (8 sujetos), intermedio (24 sujetos) y junior (42 sujetos). Los 42 sujetos que participan en las dos primeras fases del experimento fueron miembros del curso de nivel superior "Diseño y Desarrollo de Software" de la Universidad de Maryland en 1982 y 1983. Los participantes poseen un nivel superior en informática, algunos trabajaban a tiempo parcial, y todos están en un buen nivel académico. Los 32 sujetos de la tercer fase son profesionales de programación de la NASA y de Sciences Corporation. Estas personas son matemáticos, físicos, e ingenieros que desarrollan software de apoyo en tierra para los satélites. Ellos están familiarizados con las tres técnicas de prueba.

Los programas utilizados en el experimento son seleccionados especialmente y se presentan a los sujetos para las capacitaciones, los sujetos no verifican programas escritos por ellos. Todos los programas son codificados en un lenguaje de alto nivel con el que los sujetos están familiarizados. Los mismos no tienen comentarios, por lo que se dificulta la inspección con la técnica de revisión de código. Los tres programas probados en la fase 3 están programados en Fortran, mientras que los probados en las fases 1 y 2 están programados en Simpl-T.

Los cuatro programas probados son: un procesador de texto (P1), una rutina matemática (P2), un tipo de datos abstracto numérico (P3), y un mantenedor de base de datos (P4). Cada sujeto verifica 3 programas, pero un total de 4 programas se usan durante las 3 fases del experimento. El tamaño de los programas en líneas de código corresponde a 169, 145, 147 y 365, respectivamente. Algunos programas tienen solamente las faltas que se cometieron durante su desarrollo, y otros tienen una combinación de las faltas originales y faltas sembradas. Entre los cuatro programas hay un total de 34 faltas: P1 tiene 9, P2 tiene 6, P3 tiene 7, y P4 tiene 12.

Las faltas en los programas se clasifican de acuerdo a dos sistemas de clasificación. El primero clasifica las faltas en faltas de omisión y faltas de comisión. Faltas de omisión son las originadas como resultado de olvidar alguna entidad por parte del programador. Faltas de comisión son las que se producen como

consecuencia de un segmento de código incorrecto. La segunda categorización de faltas consiste en una partición de las faltas en 6 clases: 1) inicialización, 2) cálculo, 3) control, 4) interfaz, 5) datos, y 6) cosméticas.

El diseño elegido posee tres factores: la técnica, el programa, y el nivel de experiencia de los sujetos. El diseño experimental aplicado a cada fase del estudio es un diseño factorial fraccional. En las primeras dos fases solo participan sujetos de los niveles de experiencia intermedio y junior, mientras que en la tercer fase participan sujetos de los tres niveles. Por lo tanto, solo en la tercer fase sujetos de los tres niveles de experiencia aplican las tres técnicas. En cada fase se utilizan tres de los cuatro programas: en la fase uno se verifican los programas P1, P2 y P3, en la fase dos los programas P1, P2 y P4, mientras que en la fase tres los programas P1, P3 y P4.

En cada fase todos los sujetos utilizan las tres técnicas y verifican los tres programas participantes de la fase. Además, todas las combinaciones de nivel de experiencia, técnica y programa ocurren en las tres fases (considerando los niveles de experiencia y programas de cada fase). Ningún sujeto verifica un mismo programa más de una vez. El orden de aplicación de las técnicas en cada fase fue asignado de forma aleatoria a los sujetos de cada nivel de experiencia.

Cada fase se compone de 5 sesiones: una capacitación inicial, tres sesiones de verificación del software, y una sesión de seguimiento. Todos los sujetos son expuestos a clases similares de capacitación sobre las técnicas a utilizar durante el experimento. Las prácticas son realizadas en días distintos a causa de la cantidad de esfuerzo requerido. Sin embargo, dentro de cada fase todos los sujetos verifican el mismo programa en el mismo día.

El experimento consta de un conjunto de variables de respuesta, siendo las principales: el número y porcentaje de faltas detectadas, el tiempo total de detección, y tasa de detección de faltas. Para las técnicas de testing funcional y testing estructural también se miden las siguientes variables: número de ejecuciones, tiempo de CPU consumido, máxima cobertura de sentencias lograda, tiempo de conexión utilizado, número y porcentaje de faltas observables a partir de los datos, y porcentaje de faltas observables a partir de los datos que efectivamente fueron observadas por los sujetos.

Del experimento realizado resultan varias observaciones. Los sujetos del nivel de experiencia avanzado detectaron más faltas y fueron más eficientes utilizando revisión de código que utilizando las técnicas de testing estructural y testing funcional. Además, detectaron más faltas con testing funcional que con estructural. Los sujetos pertenecientes a los niveles intermedio y junior fueron aproximadamente iguales de eficaces y eficientes con las tres técnicas. La eficacia, eficiencia y costo dependen del tipo de software verificado. La técnica de revisión de código detecta más faltas de interfaz que las otras técnicas, mientras que el testing funcional detecta más faltas de control. Los sujetos estimaron mejor el porcentaje de faltas detectadas cuando aplicaron revisión de código, y peor cuando utilizaron testing funcional. Como conclusión general se observa que la técnica de revisión de código es tan efectiva como las técnicas de testing estructural y testing funcional en cuanto al número de faltas detectadas y costo asociado.

## 3.2. Experimento Kamsties - Lott (1995)

El experimento de Kamsties-Lott se basa en el experimento llevado a cabo en 1987 por Basili y Selby, y tiene como objetivo evaluar la eficiencia y eficacia de tres técnicas de verificación en la detección de faltas.

El propósito del experimento es aportar a la evidencia empírica existente, como apoyo a la toma de decisiones de los desarrolladores referente a las técnicas de verificación a utilizar en determinadas condiciones.

Las técnicas utilizadas corresponden a revisión de código, testing funcional y testing estructural. Dentro del testing funcional se utilizan clases de equivalencia y valores límite, y para testing estructural los criterios de cobertura de ramificaciones, ciclos, y operadores relacionales.

La hipótesis del experimento dice que las técnicas mencionadas anteriormente difieren en efectividad y eficiencia, y a su vez, difieren en efectividad en la detección de faltas de diferentes tipos. Como hipótesis complementaria se plantea que la motivación y habilidades de los sujetos predicen la efectividad y eficiencia de los mismos.

Se extiende el experimento de Basili-Selby al incorporar el paso de detectar las faltas luego de ser observadas las fallas. Esto implica que los sujetos deben identificar en el código la falta que provoca la falla observada.

El diseño elegido es un diseño fraccional factorial de $3^2$ x 6, el cual se encuentra resumido en el Cuadro 3.1. El diseño consiste en que cada sujeto aplique cada técnica a un programa diferente, aplicando todos la misma técnica a los mismos programas. Los sujetos son repartidos en seis grupos. Todos los grupos verifican el mismo programa en el mismo día, utilizando cada grupo una única técnica para cada programa. Este diseño resulta fraccional porque los sujetos no aplican todas las técnicas a todos los programas.

Se consideran tres factores: las técnicas de verificación, los programas sobre los cuales se aplican las técnicas, y el orden en que se aplican las mismas. Como ya fue mencionado, el primer factor presenta tres alternativas (revisión de código, testing funcional y testing estructural). El segundo factor tiene también tres alternativas (tres pequeños programas en lenguaje C), mientras que el tercer factor posee seis alternativas, que corresponden a todos los órdenes posibles de ejecución de las técnicas.

Cuadro 3.1: Diseño utilizado por Kamsties y Lott.

| Programa y día | Revisión de código | Testing funcional | Testing estructural |
|---|---|---|---|
| prog 1 (día 1) | grupos 1, 2 | grupos 3,4 | grupos 5, 6 |
| prog 2 (día 2) | grupos 3, 5 | grupos 1, 6 | grupos 2, 4 |
| prog 3 (día 3) | grupos 4, 6 | grupos 2, 5 | grupos 1, 3 |

La aleatorización consiste en asignar los sujetos al azar a uno de los grupos. De esta manera quedan determinadas las combinaciones de técnica - programa asignadas a cada sujeto, así como también el orden en que aplica las técnicas. Con el diseño elegido se asegura que cada sujeto aplique las tres técnicas y

trabaje con los tres programas, pero una única vez, es decir que para un mismo sujeto no se repite la técnica ni el programa en los experimentos unitarios en los que participa. También se aseguran todas las combinaciones de técnicas y programas (con la participación de diferentes sujetos), y todos los órdenes posibles de aplicación de las técnicas (sobre diferentes programas).

El hecho de que los sujetos trabajen con cada técnica y programa una única vez minimiza los efectos de aprendizaje. A esto se agrega la variación del orden de aplicación de las técnicas, lo cual reparte el efecto causado por el orden entre todos los sujetos. La aleatorización de los sujetos distribuye los efectos causados por sus diferencias de habilidades entre todas las técnicas. Con la utilización de todas las combinaciones técnica - programa se reparten los efectos causados por las diferencias de los programas entre todas las técnicas.

Otro aspecto a considerar es que las técnicas y las guías seguidas son reconocidas por los investigadores como representativas de la práctica de la industria. Las guías utilizadas fueron diseñadas por Basili y Selby con dicho fin. Además, la incorporación del paso de detectar las faltas luego de observadas las fallas es considerada por Kamsties-Lott como una extensión que hace al experimento más representativo.

Antes de realizarse el experimento se presentan las técnicas de verificación a los sujetos y los mismos se ejercitan mediante una práctica. Los sujetos poseen algo de experiencia con el lenguaje C. Para las prácticas se utilizan tres programas codificados en lenguaje C, con aproximadamente la mitad del tamaño de los programas verificados en el experimento.

Los programas utilizados en el experimento consisten en un conjunto de funciones (de 10 a 30 líneas) contenidas en un único archivo, con el correspondiente archivo de cabezal (de unas 30 líneas). Los programas son casi del mismo largo y utilizan estructuras de control similares. La mayoría de las faltas presentes en los programas fueron sembradas en el código y todas producen fallas, no hay cancelación entre las mismas.

El esquema de clasificación utilizado es el propuesto por Basili y Selby en su experimento. Las faltas se clasifican en un primer nivel entre omisión (ausencia de código necesario) y comisión (presencia de código incorrecto) , mientras que en un segundo nivel se distingue la clase de la falta (inicialización, cómputo, control, interfaz, datos, cosmética).

El experimento consiste en dos réplicas internas, las cuales son realizadas en tres días fijados en una semana. Se utilizaron las mismas guías que en las prácticas de ejercitación.

Los sujetos que participan del experimento son estudiantes universitarios que pertenecen a cursos de Ingeniería de Software y todos transitan al menos su tercer año de carrera. En la primer réplica participan 27 estudiantes durante todo el transcurso de la misma, mientras que en la segunda réplica el número de estudiantes fue disminuyendo con el paso de los días, siendo de 23, 19 y 15. No existen sujetos en común entre las réplicas.

Los autores distinguen entre fallas no reveladas, fallas reveladas y fallas observadas. En las técnicas de testing funcional y testing estructural las fallas no reveladas son aquellas para las cuales no se diseñó un caso de prueba que las manifieste. Las fallas reveladas corresponden a las fallas que son manifestadas a partir de algún caso de prueba diseñado, mientras que las fallas observadas son las fallas que los sujetos efectivamente perciben. Es decir, una falla es revelada pero no observada si el sujeto no se entera de su existencia. En la revisión de

código una falla es revelada si el sujeto captura la inconsistencia en su propia especificación del programa, y observada si visualiza la inconsistencia entre su especificación y la proporcionada (real).

En el caso de las faltas los autores distinguen entre faltas no detectadas y faltas detectadas. Para las tres técnicas, una falta es detectada si el sujeto logra describir el problema existente en el código con suficiente precisión. Una falta es no detectada en caso contrario.

Las variables de respuesta consideradas corresponden a la cantidad de fallas y faltas pertenecientes a cada categoría de las explicadas anteriormente, así como al tiempo invertido en cada paso, entre otras. Para determinar los valores de las mismas se considera información brindada por los sujetos mediante el llenado de formularios acerca de su motivación y habilidades, y principalmente, los registros de las fallas observadas y faltas detectadas, así como el tiempo insumido por cada actividad.

Las métricas consideradas son:

- Motivación en cada día (estimación subjetiva, 0..5).
- Habilidades con el lenguaje C (estimación subjetiva, 0..5).
- Tiempo que han trabajado con C (años).
- Habilidades aplicando las técnicas de detección de defectos (estimación subjetiva, 0..5).
- Tiempo invertido en cada paso (minutos).
- Número de fallas reveladas (cantidad).
- Número de fallas observadas (cantidad).
- Número total de faltas aisladas (cantidad).
- Número de faltas aisladas puramente por casualidad (cantidad).
- Número de faltas aisladas utilizando las técnicas (total menos casualidad; cantidad).

De los resultados obtenidos en el experimento, los autores concluyen que: bajo las condiciones de sujetos relativamente inexperientes con el lenguaje y con las tres técnicas estudiadas, si no se considera al tiempo como un punto de interés, cualquiera de las técnicas puede ser utilizada con igual efectividad en la detección de faltas. Es decir, sujetos inexpertos pueden aplicar una técnica de verificación formal (revisión de código) con igual efectividad que una técnica basada en la ejecución. Sin embargo, fueron más eficientes al aplicar testing funcional. Las diferencias de efectividad observadas entre las técnicas con respecto a las clases de tipos de faltas, sugieren que una combinación de técnicas puede superar el desempeño de una técnica aislada.

## 3.3. Experimento F. Macdonald y J. Miller (1997)

El experimento de Macdonald y Miler compara la efectividad de la técnica de revisión de código con dos métodos diferentes: inspección basada en papel e inspección basada en una herramienta. La herramienta utilizada es un prototipo llamado ASSIST (Asynchronous/Synchronous Software Inspection Support Tool).

La inspección basada en papel es una técnica muy efectiva para la detección de faltas. La experiencia indica que el rendimiento de la inversión por cada hora de inspección provoca una disminución de 33 horas de mantenimiento posterior. A pesar de los beneficios, la inspección sigue siendo un proceso muy caro. Este gasto se debe principalmente a la cantidad de sujetos que están involucrados y la cantidad de tiempo que se gasta, tanto individual como grupal.

La herramienta ASSIST permite a los sujetos mantener en línea los documentos de las inspecciones, soportando la asociación de anotaciones a los mismos, además de brindar apoyo en las reuniones grupales. La herramienta provee una ventana de ejecución que permite conocer la lista de usuarios junto con sus estados, además de la lista de archivos accesibles. Cuenta con un *browser* que permite a los sujetos manipular una lista de ítems, donde realizan las anotaciones sobre los documentos inspeccionados. Los ítems pertenecientes a las listas individuales pueden unirse en una lista general durante las inspecciones grupales. Provee también un browser para ver el texto de los documentos a inspeccionar, basándose en el concepto de "posición actual", permitiendo realizar nuevas anotaciones referentes a la línea actual o leer las ya ingresadas para la misma.

El objetivo del experimento es averiguar si existen diferencias de eficiencia entre las inspecciones basadas en papel y las inspecciones basadas en la herramienta, comparando el número de faltas detectadas en un período de tiempo determinado, tanto para los sujetos trabajando en forma individual como en grupo.

El experimento es realizado por dos grupos de sujetos que inspeccionan un único programa, uno de los cuales utiliza la técnica basada en papel y el otro grupo utiliza la herramienta. Para asegurar que cualquier falta encontrada por un grupo no se debe a la capacidad superior del mismo, se realiza una segunda inspección en la cual verifican otro programa con la técnica alternativa.

El proceso de inspección se realiza en 2 etapas. La primer etapa es de detección individual, donde cada sujeto busca las faltas del programa, y la segunda etapa es de reunión grupal, donde se realiza la consolidación de las listas de faltas encontradas individualmente en una sola lista.

En total participan 43 sujetos, que utilizan ASSIST y papel para inspeccionar 2 programas escritos en C++ de aproximadamente 150 líneas de código. Los sujetos son estudiantes de tercer año de universidad, y tienen una base sólida de programación en Scheme, C++ y Eiffel. Además, realizaron un curso de Introducción a la Ingeniería de Software. Poseen una alta motivación en el experimento ya que es parte de un curso y tiene nota asociada.

Los sujetos que participan del experimento son divididos en dos sesiones aproximadamente iguales. La primer sesión de 22 sujetos y la segunda de 21. Los sujetos fueron repartidos en las dos sesiones basándose en su desempeño con C++, de forma de que los miembros de ambos grupos tengan habilidades similares. Dentro de las dos sesiones, los sujetos se organizaron en grupos de

tres (y un único grupo de cuatro), creando grupos equitativos en cuanto a las habilidades de programación en C++ de sus integrantes. La sesión 1 consiste en seis grupos de tres sujetos y un grupo de cuatro, mientras que la sesión 2 contiene siete grupos de tres sujetos. Todos los grupos inspeccionan ambos programas en el mismo orden. Cada sujeto aplica un único método a cada programa. Cada inspección se realiza en el transcurso de dos semanas, en la primer semana los sujetos efectúan las inspecciones individuales, mientras que en la segunda se realizan las reuniones grupales.

En el Cuadro 3.2 se detalla como es el diseño del experimento.

Cuadro 3.2: Diseño utilizado por Macdonald y Miller en el Experimento.

| Programa y semana | Inspección basada en papel | Inspección basada en la herramienta |
|---|---|---|
| Prog 1 (7 y 8) | Sesión 2 (grupos 8 al 14) | Sesión 1 (grupos 1 al 7) |
| Prog 2 (9 y 10) | Sesión 1 (grupos 1 al 7) | Sesión 2 (grupos 8 al 14) |

Para la etapa de entrenamiento se utilizaron algunos programas provenientes del experimento de Kamsties y Lott. Dichos programas fueron traducidos a C++ y modificados para eliminar algunas faltas e introducir nuevas . Dado que parte del material proveniente del experimento de Kamsties y Lott fue utilizado un año antes por la misma clase, los programas elegidos para el experimento fueron escritos nuevamente. Los mismos constan de 147 y 143 líneas de código, con 12 faltas cada uno. A éstos se agrega una especificación y una lista de funciones de librería, manteniendo la similitud con el estilo de los materiales de Kamsties y Lott.

Cada sesión práctica fue limitada a 2 horas de duración, dando una tasa de inspección de unas 70 líneas por hora. Para las inspecciones basadas en papel los sujetos completaron reportes de faltas detectadas, individuales y grupales. Para la inspección basada en ASSIST utilizaron listas de faltas individuales, y una lista general para la inspección en grupo.

Por cada sujeto y cada grupo se registra el número de faltas detectados correctamente y el número de “falsos positivos”. Los “falsos positivos” consisten en ítems reportadas como faltas, cuando en realidad no existe falta, constituyendo una equivocación por parte del sujeto. Se consideran también las ganancias y pérdidas en las inspecciones grupales. Las pérdidas corresponden a las faltas que son reportados por algún integrante en forma individual, pero no por el grupo. Las ganancias corresponden a las faltas que no son reportados por ningún integrante en forma individual, pero si por el grupo. En base a los datos obtenidos en los reportes se calcula la frecuencia de detección para cada falta, tanto para las inspecciones basadas en papel como para las basadas en la herramienta.

El experimento se realiza durante un período de diez semanas. En las primeras seis semanas se proporciona a los sujetos formación en inspección de software y uso de ASSIST, así como actualizar sus conocimientos de C++. En las restantes cuatro semanas se lleva a cabo el experimento.

La inspección propiamente dicha consiste en utilizar la especificación del programa y la lista de funciones de librería para inspeccionar el código fuente, haciendo uso de una lista de verificación.

Del experimento resulta que no hay diferencia significativa de eficiencia entre las inspecciones basadas en papel y las basadas en la herramienta, ya sea para los

sujetos trabajando en forma individual como en grupo. La eficiencia es medida en términos del número de faltas detectadas en un período de tiempo determinado.

El número de falsos positivos reportados, y el costo ganado o perdido a causa de las reuniones grupales también es considerado, y aquí tampoco se presentan diferencias significativas entre ambos métodos de inspección.

## 3.4. Experimento Natalia Juristo y Sira Vegas (2001)

El experimento realizado por Natalia Juristo y Sira Vega se basa también en el realizado por Basili y Selby. El objetivo del mismo es estudiar la eficacia y eficiencia de diferentes técnicas de verificación. Para esto se investiga si la eficacia de la técnica depende o no del tipo de falta. El estudio compara la efectividad relativa de diferentes técnicas de verificación y las relaciona con el tipo de falla detectada.

Para probar si la eficacia de las técnicas está relacionada con el tipo de faltas en el programa, se mide la eficacia en términos de la cantidad de sujetos que detectaron una falta para cada falta en el programa.

El experimento tiene como factores la técnica que se utiliza, el tipo de falta, y el programa. Todos los programas tienen la misma cantidad de faltas, y las mismas se clasifican usando la clasificación descrita por Basili.

Las técnicas que se utilizan son testing funcional, testing estructural, y revisión de código. El testing funcional se basa en partición de clases de equivalencia, y para la ejecución del mismo se entrega tanto el código fuente como la especificación. Para el testing estructural se facilita el código fuente sin especificación. Los sujetos escriben sus casos de prueba y resultados obtenidos, y luego se les entrega la especificación para poder probar la exactitud de los resultados. El criterio que se usa en testing estructural no queda del todo claro, el artículo plantea como criterio lo más cercano posible a cubrimiento de decisión. Para la revisión de código se suministra el programa en texto plano , se solicita que se identifiquen rutinas, subrutinas y se escriba su especificación. Una vez terminada la especificación se compara con la especificación original.

Se utilizan cuatro programas diferentes, dos programas muy similares de cada tipo de software (2 de datos y 2 funcionales).

Los parámetros del experimento son los siguientes:

- El tamaño del programa: El tamaño del programa es similar al de los programas propuestos por Basili. Contienen aproximadamente 200 líneas de código, con exclusión de líneas en blanco y comentarios.

- Sujetos: Sujetos de quinto año de la Escuela de Ciencias de la Computación, Universidad Politécnica de Madrid, plan de 1983. Se trata de sujetos con muy poca experiencia.

- Lenguaje de programación utilizado: El lenguaje de programación que se utiliza es C.

- Faltas: Cada programa tiene el mismo número de faltas del mismo tipo. Son un total de 9 faltas, 6 de tipo comisión y 3 de tipo omisión.

Para realizar la experiencia los sujetos se dividen en grupos, donde cada grupo representa un conjunto de sujetos que realizan el experimento (individualmente) al mismo tiempo, utilizando el mismo programa y aplicando la misma técnica. El total de sujetos es 196, dividido en 8 grupos de 12 (de los cuales 4 de los grupos prueban con técnicas estructurales y 4 con técnicas funcionales) y 4 grupos de 25 (probando con revisión de código). Todos los sujetos son conscientes de que están siendo parte de un experimento.

El experimento es organizado en 5 sesiones. Una sesión inicial de aprendizaje y 4 sesiones posteriores. Cada sesión se ejecuta en un día, donde se prueba un programa por 3 grupos que ejecutan cada uno una técnica distinta de las 3 posibles.

Con el fin de maximizar la aleatoriedad del experimento, el procedimiento para la asignación de días, programas y técnicas a los grupos es el siguiente: Se coloca en una bolsa papeles que contienen los diferentes grupos (del 1 al 12). Se van sacando papeles de a 3 para conformar lotes, así se obtienen 4 lotes de 3 grupos cada uno. A posterior, por cada lote se saca un papel de una bolsa que contiene los días (del 1 al 4), y uno de la bolsa que contiene los programas (P1, P2, P3 y P4) hasta que todos los lotes tengan día y programa asignados. Después de esto se elaboró una lista de los grupos en orden creciente y se colocó en una bolsa 3 papeles que corresponden a las técnicas. Se fue sacando técnicas de la bolsa y asignando a los grupos en orden hasta que la bolsa queda vacía, luego se colocan nuevamente en ella las técnicas para seguir determinado las técnicas a los grupos faltantes.

En el Cuadro 3.3 se detalla cómo es el diseño del experimento.

Cuadro 3.3: Diseño utilizado por Juristo y Sira en el Experimento I.

| Programa y día | Revisión de código | Testing funcional | Testing estructural |
|---|---|---|---|
| Prog 1 (día 1) | grupo 6 | grupo 5 | grupo 9 |
| Prog 2 (día 2) | grupo 3 | grupo 1 | grupo 10 |
| Prog 3 (día 3) | grupo 8 | grupo 2 | grupo 4 |
| Prog 4 (día 4) | grupo 12 | grupo 7 | grupo 11 |

Con la asignación de una única técnica a un único programa para cada grupo se elimina la amenaza de aprendizaje obtenido. También se elimina la influencia de características individuales, dado que todos los sujetos tienen la misma experiencia y con perfiles bastante homogéneos. Las faltas son inyectadas pero se trata de que estas representen las faltas reales cometidas por los programadores.

Como se mencionó anteriormente, la variable de respuesta es el número de sujetos que han detectado cada falta que contiene el programa.

Algunas de las conclusiones que se extraen son que el número de sujetos que detecta una falta depende del programa que se está probando, la técnica utilizada y la falta. Además, existen faltas que se comportan mejor para ciertos programas y faltas que se detectan mejor con el uso de ciertas técnicas.

El hecho de que una falta se comporta mejor con un programa en particular

se interpreta en el sentido de que la falta no puede ser considerada de forma independiente del programa en el que la misma se produce.

Además, es interesante observar que el hecho de que un sujeto no observe la falla correspondiente a una falta se debe a dos razones muy diferentes:

- No se genera un caso de prueba que revele la falla.
- La prueba fue generada, pero el sujeto no puede ver el fracaso en la pantalla.

En los datos recogidos utilizando ANOVA se observa que la técnica de revisión de código es menos sensible a ocultar faltas que las otras técnicas. A pesar de que la técnica funcional se comporta mejor que la técnica estructural en la mayoría de los casos, los casos en que las dos técnicas se comportan de forma idéntica, o una mejor que la otra, se producen indistintamente para cada tipo de falta.

Como los sujetos ejecutan sus propios casos de prueba, y la variable de respuesta mide el número de sujetos que detectan la falla producida por la falta, es imposible detectar si es la técnica la que no genera un caso de prueba que presenta la falla o es el sujeto el que no observa la falla. Esto lleva a considerar examinar la capacidad de detectar la falta y la visibilidad de la falla en el experimento II.

En este experimento no se tuvo en cuenta que la asignación de sujetos al azar puede funcionar incorrectamente. Estudios previos a éste ya descubrieron que los sujetos que aplican las mismas técnicas no encuentran generalmente las mismas faltas. Esto lleva a considerar la posibilidad de modificar el diseño del experimento para el Experimento II, para que todos los sujetos apliquen todas las técnicas.

Basándose en las conclusiones del primer experimento se realiza un segundo experimento, con el propósito de poder llegar a conclusiones que no pueden obtenerse en el Experimento I debido a las limitaciones del diseño.

El objetivo del segundo experimento es investigar tres aspectos:

1. **Influencia de la falta en la visibilidad:** Como no se puede deducir de la experiencia anterior si el hecho de que un sujeto no detecta una falla se debe a que la técnica no logró producir un caso de prueba que revele la misma o porque el sujeto no observa la falla cuando se produce, se estudia cómo es la visibilidad de las fallas causados por las faltas. En el experimento anterior los sujetos ejecutan los casos de prueba que generan para detectar las posibles faltas del programa. En este experimento se le solicita a los sujetos que ejecuten un conjunto de casos de prueba ya generado, donde todas las faltas se detectan en el programa.

2. **Influencia de la técnica y el tipo de falta:** Se investiga en qué medida el uso de una o varias técnicas de detección influye en el tipo de falta detectada.

3. **Influencia de la posición de la falta:** En el experimento anterior se descubre que prácticamente no hay diferencia entre el número de sujetos que detectan cada falta utilizando la técnica de revisión de código. Se busca entonces otra característica de la falta que puede influir en la eficacia, para esto se investiga la posición de la falta en el programa.

Como el tamaño de los programas no es muy grande, no es posible insertar muchas faltas, por lo que se opta por manejar dos versiones de cada programa. Esto da lugar a la replicación de faltas en los programas. La versión se introduce como un nuevo factor en el experimento.

Los parámetros son los mismos que en el experimento anterior. Se agrega la versión a cada programa, y cada programa tiene ahora 7 faltas. Las versiones de los programas contienen el mismo número de faltas y son del mismo tipo.

La variable de respuesta para este experimento cambió y se define como el número de sujetos que generan un caso de prueba capaz de detectar la falta.

En esta experiencia los sujetos son 46 y se dividen en 6 grupos de 7 a 8 sujetos. El diseño de este experimento cambió, en este caso todos los grupos ejecutan todas las técnicas.

En el Cuadro 3.4 se detalla como es el diseño del experimento.

Cuadro 3.4: Diseño utilizado por Juristo y Sira en el Experimento II.

| Programa y día | Revisión de código | Testing funcional | Testing estructural |
|---|---|---|---|
| Prog 1 (día 1) | grupos 1, 2 | grupos 5, 6 | grupos 3, 4 |
| Prog 2 (día 2) | grupos 4, 5 | grupos 2, 3 | grupos 1, 6 |
| Prog 3 (día 3) | grupos 3, 6 | grupos 1, 4 | grupos 2, 5 |

Las conclusiones se obtienen realizando una comparación entre los resultados de los Experimentos I y II. Algunas conclusiones que se extraen de ANOVA son que el número de sujetos que generan un caso de prueba para la detección de una falta depende de la versión, la técnica utilizada, y la falta en cuestión. Además, existen faltas que se comportan mejor para ciertos programas, faltas que se detectan más con el uso de ciertas técnicas, y programas que se comportan mejor para ciertas técnicas.

En comparación con el Experimento I, es de destacar que el efecto del programa no llega a ser importante en este caso, mientras que la versión y la combinación programa/técnica sí. El ANOVA muestra la interacción entre la técnica y la falta en este experimento, como lo hace en el Experimento I. Como se sospecha en el Experimento I, y se muestra mucho más claramente en este, la revisión de código siempre se comporta peor que las técnicas funcionales y estructurales, independientemente de la falta.

En cuanto a la comparación entre las técnicas funcionales y estructurales, en el experimento I se comporta mejor la técnica funcional que la técnica estructural en la mayoría de los casos. En el experimento II se encuentra que las dos técnicas se comportan igual. Esto puede atribuirse al hecho de que ahora se está estudiando si la técnica genera casos de prueba que detectan un tipo de falta específico. A diferencia del Experimento I, donde no hay técnicas que mejor se comporten según el programa, aquí se encuentra que el programa determina el comportamiento de la técnica, aunque en general afirma que las técnicas estructurales y funcionales se comportan de manera similar y mejor que la revisión de código.

Desde el ANOVA se deduce que ninguno de los parámetros considerados y

ninguna de sus combinaciones influye en los resultados y, por lo tanto, el número de sujetos que observa una falla utilizando la técnica de revisión no depende de la visibilidad de la falta. Esto significa que la revisión de código no tiene preferencia por los tipos específicos de faltas. Este es el mismo resultado que se encuentra para las técnicas funcionales y estructurales, que se comportan igual para los diferentes tipos de defectos.

Con respecto a la influencia de la versión, este experimento dió un resultado que no se esperaba, indicando que la versión influye en el número de sujetos que detectan una falta. Independientemente del programa, la técnica y la falta, más sujetos generan casos de prueba que detectan las faltas en una versión y menos en otra versión. Con este resultado se comprueba la observación realizada en el Experimento I en relación a la adecuación de la clasificación de faltas utilizada, y la necesidad de un esquema de clasificación de faltas sensibles a la técnica.

## 3.5. Marco de Comparación

En esta sección planteamos las características que consideramos relevantes para realizar una comparación de experimentos formales que evalúan técnicas de verificación. En primer lugar es de interés comparar los objetivos. De esta manera se puede conocer hasta qué punto y en qué aspectos los experimentos son comparables. En segundo lugar identificar los factores establecidos para el experimento y las alternativas para cada uno de ellos. Se comparan también dichas alternativas y se distinguen los parámetros fijados en cada experimento.

Un componente fundamental de los experimentos en Ingeniería de Software son los sujetos, por lo cual nos resulta interesante comparar sus principales características, como son su experiencia, habilidades y motivación.

Dado que los experimentos a ser comparados refieren a la aplicación de técnicas de verificación, es relevante comparar aspectos propios de este tipo de experimentos como ser la clasificación de faltas utilizada, el número de faltas presentes en cada programa, tamaño y lenguaje de los programas, etc.

Es de primordial importancia realizar una comparación centrada en el diseño elegido para cada experimento. Esto comprende la duración del experimento, la distribución de los sujetos, los lineamientos seguidos para la asignación de las combinaciones de técnica y programa a los sujetos, la división del experimento en sesiones, la cantidad de experimentos unitarios en los que participa cada sujeto, etc.

Realizamos también una comparación en la forma en la que se aplica cada diseño, es decir, el proceso seguido en cada experimento. Identificamos las variables de respuesta elegidas por los autores de los diferentes experimentos. Por último, vemos las diferencias y similitudes en las conclusiones alcanzadas en cada experimento.

## 3.6. Comparación de los Artículos

Realizamos un análisis comparativo de los experimentos presentados anteriormente aplicando el marco de comparación. De aquí en adelante nos referiremos a los autores como: Basili y Selby (B-S), Kamsties y Lott (K-L), F. Macdonald y J. Miller (M-M), y Natalia Juristo y Sira Vegas (J-V).

Todos los experimentos tienen en común sus objetivos, los cuáles son evaluar la eficiencia y eficacia de técnicas de verificación para la detección de faltas. Para realizar esta evaluación se realizan estudios sometiendo sujetos a la ejecución de técnicas de verificación sobre programas o fracciones de código.

B-S son los primeros en realizar el experimento en 1987, posteriormente K-L en 1995 realizan su experimento en base al anterior, pero no es una réplica ya que difiere en cuanto al lenguaje de programación utilizado y el proceso de verificación al incorporar el paso de detectar las faltas luego de ser observadas las fallas. En 1997 M-M realizan un experimento, pero tiene variaciones con respecto a los anteriores que iremos detallando en esta sección. Finalmente J-V en el 2001 realizan un primer experimento basándose en los experimentos de B-S y K-L, y un segundo basándose en los mismos experimentos anteriores además de su experiencia adquirida.

Los factores considerados en cada experimento pueden observarse en el Cuadro 3.5.

Cuadro 3.5: Factores

| Factor | B-S | K-L | M-M | J-V(1) | J-V(2) |
|---|---|---|---|---|---|
| Técnica | √ | √ | | √ | √ |
| Programa | √ | √ | √ | √ | √ |
| Experiencia de sujetos | √ | | | | |
| Orden de aplicación | | √ | | | |
| Tipo de falta | | | | √ | √ |
| Método de inspección | N/A | N/A | √ | N/A | N/A |
| Versión del programa | N/A | N/A | N/A | N/A | √ |

Los estudios realizados por B-S, K-L y J-V utilizan las mismas técnicas de verificación como alternativas al factor técnica, que son: revisión de código, testing funcional y testing estructural. Los autores M-M utilizan la técnica de revisión de código con dos métodos diferentes: inspecciones basadas en papel y basadas en herramientas de software. Todos tienen en común el programa como factor. Cada experimento tiene también factores que difieren de los demás, B-S considera como factor la experiencia de los sujetos, K-L el orden en que se aplican las técnicas, M-M el método de inspección, J-V el tipo de falta en ambos experimentos, y en el último agrega la versión del programa. El método de inspección considerado por M-M y la versión del programa considerada por J-V son aspectos que tienen sentido solo para dichos experimentos, ya que en los restantes experimentos no se utilizan diferentes métodos de aplicación de las técnicas ni diferentes versiones de los programas probados.

Los parámetros considerados en cada experimento se muestran en el Cuadro 3.6.

Cuadro 3.6: Parámetros

| Parámetro | B-S | K-L | M-M | J-V(1) | J-V(2) |
|---|---|---|---|---|---|
| Lenguaje | √ | √ | √ | √ | √ |
| Tamaño de los programas | √ | √ | √ | √ | √ |
| Faltas | √ | √ | √ | √ | √ |
| Sujetos | | √ | √ | √ | √ |

Todos los experimentos establecen sus parámetros, como son el tamaño de los programas, el lenguaje en que estos son programados y las faltas que contienen. Los sujetos son considerados parámetro por K-L, J-V y M-M, mientras que para B-S los mismos constituyen un factor, ya que varía el nivel de experiencia en su diseño.

Las características de los programas utilizados en los experimentos se detallan en el Cuadro 3.7.

Cuadro 3.7: Características de los programas

| Características | B-S | K-L | M-M | J-V(1) | J-V(2) |
|---|---|---|---|---|---|
| Cant. de programas | 4 | 3 | 2 | 4 | 3 |
| Tamaño de programas | 169, 145, 147 y 365 LOCs c/prog | cjto de funciones de 10 a 30 LOCs | 147 y 143 LOCs c/prog | 200 LOCs | 200 |
| Cantidad de faltas | 34 en total | no especifica | 12 c/prog | 9 c/prog | 7 c/prog |
| Lenguaje | Fortran y Simpl-T | C | C++ | C | C |

Los programas a probar en todos los experimentos son considerados chicos (contienen menos de 500 LOCs), el lenguaje de desarrollo utilizado es C por K-L y J-V, C++ por M-M, y Fortran y Simpl-T por B-S. Las faltas que estos contienen son para M-M 12 faltas y para J-V 9 y 7 faltas para sus experimentos uno y dos respectivamente. En dichos experimentos el número de faltas es igual para todos los programas. Los cuatro programas utilizados por B-S difieren en la cantidad de faltas que contienen, en total son 34, distribuyéndose de la siguiente manera: el procesador de texto contiene 9, la rutina matemática contiene 6, el tipo de datos abstracto numérico contiene 7, y el mantenedor de base de datos contiene 12. En el caso de K-L no se especifica el número de faltas.

Las principales características de los sujetos se presentan en el Cuadro 3.8.

Los sujetos que realizan las revisiones tienen diferentes niveles de estudios y experiencia. En el experimento de B-S se eligen los sujetos para ser representativos de los diferentes niveles de conocimientos de la realidad, habiendo niveles avanzado, intermedio y junior, existiendo entre éstos miembros del curso de nivel superior “Diseño y Desarrollo de Software” de la Universidad de Maryland, profesionales de programación de la NASA y de Sciences Corporation. En el experimento de K-L los sujetos son estudiantes de tercer y cuarto año de la Universidad de Kaiserslautern. Los mismos poseen algo de experiencia en C, pero de todos modos se realiza una práctica previa para que se ejerciten con el uso de las técnicas y el lenguaje. Esta misma capacitación inicial es realizada en los experimentos de J-V, que eligen sujetos de quinto año de la Escuela de Ciencias de la Computación, Universidad Politécnica de Madrid con muy poca experiencia. Los sujetos del experimento de M-M son estudiantes de tercer año de universidad, y tienen una base sólida de programación en Scheme, C++ y Eiffel, poseen una alta motivación en el experimento ya que es parte de un curso y tiene nota asociada.

En en Cuadro 3.9 se muestran algunas decisiones de diseño consideradas en

Cuadro 3.8: Características de los sujetos

| Características | B-S | K-L | M-M | J-V(1) | J-V(2) |
|---|---|---|---|---|---|
| Cantidad | 74 | 50 | 43 | 196 | 46 |
| Experiencia | 8 avanzados, 24 intermedio y 42 junior | se considera un único nivel | se considera un único nivel | se considera un único nivel | se considera un único nivel |
| Nivel de estudio | estudiantes de la Universidad de Maryland, profesionales de programación de la NASA y de Sciences Corporation | estudiantes de 3er y 4to año | estudiantes de 3er año | estudiantes de 5to año | estudiantes de 5to año |

cada experimento.

Los estudios de B-S y K-L utilizaron el mismo esquema de clasificación de defectos, J-V utilizó una sub-clasificación de este esquema y M-M no clasificó los defectos encontrados. El esquema utilizado es el propuesto en el artículo de Basili, en donde las faltas se clasifican en un primer nivel en omisión y comisión. Faltas de comisión son las faltas presentes como consecuencia de un segmento de código incorrecto. Faltas de omisión son las faltas presentes como resultado de olvidar alguna entidad por parte del programador. Además se parte el sistema de categorización distinguiendo las clases de las faltas: inicialización, cálculo, control, interfaz, datos y cosméticos. Con respecto a los diseños de los diferentes autores, todos deciden realizar una capacitación previa al experimento para introducir a los sujetos con el uso de las técnicas. El experimento de B-S se divide en 5 sesiones. La primera es la capacitación, las siguientes tres son el experimento en sí mismo y posteriormente una sesión de seguimiento. Los experimentos de K-L y M-M constan de dos sesiones de experimento posterior a la capacitación. J-V organiza el experimento en 5 sesiones, la capacitación y 4 de ejecución. Durante el experimento de B-S se utilizan 4 programas en total, pero en cada sesión se prueban tres de estos, por lo que hay una combinación de programas que en dicho experimento nunca es probada. En los casos de K-L, M-M y J-V se prueban en todas las sesiones todos los programas, que son en K-L tres programas, en M-M dos y en J-V cuatro y tres respectivamente para los experimentos I y II.

En el Cuadro 3.10 se muestran características del proceso seguido en cada experimento.

Los sujetos en el experimento de B-S son 72, de los cuales 8 son avanzados,

24 de nivel intermedio y 42 junior. Los mismos se distribuyen de la siguiente manera: 29 sujetos en la primera sesión, 13 en la segunda y 32 en la tercera. En las dos primeras sesiones sólo participaron sujetos de los niveles intermedio y junior, para la tercera sesión se agregan sujetos del nivel de experiencia avanzado. En el experimento de K-L disminuye la cantidad de sujetos en el cual participan 27 sujetos en la primera réplica y 23 en la segunda. Para la segunda réplica no se repiten sujetos que participan de la primera. En el experimento de M-M la cantidad de sujetos es similar con respecto al anterior, siendo 43 participantes en total. En este caso de dividen en dos secciones de 22 y 21, siendo en la primera sección seis grupos de tres sujetos y un grupo de cuatro, mientras que en la segunda sección siete grupos de tres sujetos. Por otro lado, la cantidad de participantes aumenta en el primer experimento de J-V siendo 196 sujetos, distribuidos en 8 grupos de 12 personas (de las cuales 4 de los grupos prueban con técnicas estructurales y 4 con técnicas funcionales) y 4 grupos de 25 (probando con revisión de código). En el segundo experimento de J-V los sujetos son 46 y se dividen en 6 grupos de 7 a 8 integrantes.

El proceso seguido por Basili utiliza en cada fase tres de los cuatro programas, todos los sujetos utilizan las tres técnicas y verifican los tres programas. Dentro de cada fase todos los sujetos verifican el mismo programa el mismo día. El experimento de K-L consiste en dos réplicas internas, las cuales son realizadas en tres días fijados en una semana. En cada día se verifica un programa distinto con las tres técnicas. En el primer día participan 23 estudiantes, mientras que en el segundo 19 y en el tercero 15. El proceso de inspección de M-M se realiza en 2 etapas. La primera etapa es de detección individual y la segunda etapa es de reunión grupal donde se realiza la consolidación. El experimento se realiza durante un período de diez semanas. En las primeras seis semanas se proporciona a los alumnos formación. En las restantes cuatro semanas se lleva a cabo el experimento. La inspección propiamente dicha consiste en utilizar la especificación del programa y la lista de funciones de librería para inspeccionar el código fuente, haciendo uso de una lista de verificación. A diferencia de M-M en donde todos los sujetos aplican las 2 inspecciones y trabajan con los 2 programas, en el primer experimento de J-V cada día se lleva a cabo una sección, utilizando un programa, y 3 grupos que ejecutan cada uno, una técnica distinta de las 3 posibles. La intención de J-V con este diseño es eliminar la amenaza de aprendizaje obtenido, para K-L la intención es minimizar esta amenaza al hacer que los sujetos trabajen con cada técnica y programa una única vez. Para el experimento II de J-V cambió el diseño para que todos los sujetos ejecuten todas las técnicas. El mismo se organiza en 3 días, en donde en cada día se ejecuta un programa y todas las técnicas. Cada técnica es ejecutada por dos grupos distintos.

Las conclusiones obtenidas en cada experimento se presentan en el Cuadro 3.11.

Tanto B-S como K-L concluyen que la técnica de revisión de código es tan efectiva como las técnicas de testing estructural y funcional en cuanto al número de faltas detectadas. Además también encontraron que los sujetos fueron más eficientes al aplicar testing funcional. M-M concluye que no hay diferencia significativa de eficiencia entre las inspecciones basadas en papel y las basadas en la herramienta, ya sea para los sujetos trabajando en forma individual como en equipo. Algunas conclusiones del primer experimento de J-V indican que el número de sujetos que detecta una falta depende del programa que se está

probando, la técnica utilizada y la falta. Se observa también que la técnica de revisión de código es menos sensible a ocultar faltas que las otras técnicas. A pesar de que la técnica funcional se comporta mejor que la técnica estructural en la mayoría de los casos, los casos en que las dos técnicas que se comportan de forma idéntica o mejor que otra se producen indistintamente para cada tipo de falta. En el experimento dos concluyen que la revisión de código siempre se comporta peor que las técnicas funcionales y estructurales, independientemente de la falta. En cuanto a la comparación entre las técnicas funcionales y estructurales, encuentra que las dos técnicas se comportan igual. La inclusión de la versión en este experimento influye en el número de sujetos que detectan una falta. Independientemente del programa, la técnica y la falta, más sujetos generan casos de prueba que detectan las faltas en una versión y menos en otra versión.

## 3.7. Comentarios sobre los Artículos

En la mayoría de los casos, las faltas presentes en los programas utilizados son inyectadas en el código. Consideramos que esta práctica le resta realismo a las condiciones de ejecución del experimento, en comparación con la práctica real de la industria.

Las diferencias de habilidades y experiencia de los sujetos solo son consideradas en el experimento de B-S, dado que se considera el nivel de experiencia como un factor. En los restantes experimentos se homogeniza el nivel mediante clases niveladoras, y se eligen los estudiantes con similares características (por ejemplo que cursan el mismo año de carrera y/o asisten al mismo curso). Consideramos que la elección de B-S permite obtener información más específica respecto a la influencia de las diferencias entre los sujetos y las faltas que estos detectan. En los demás experimentos no es posible obtener información de este tipo. Es importante notar que no siempre es factible considerar los diferentes niveles de habilidades, para ello debe disponerse de los sujetos y lograr clasificarlos adecuadamente.

Un aspecto en común entre los experimentos es la intención de minimizar el riesgo de que los sujetos compartan información. Para ello aplican estrategias como ser que los sujetos trabajen con el mismo programa el mismo día.

## 3.8. Conclusiones

Se presenta en este capítulo un marco de comparación para experimentos formales que intentan observar la efectividad, costo y eficiencia de distintas técnicas de verificación. Este marco busca proveer formalidad al momento de comparar distintos experimentos. Estas comparaciones no son triviales debido a la alta variabilidad de las características de los experimentos. Además, se muestra la aplicación del marco en cuatro experimentos formales conocidos. La aplicación del marco es la comparación de cada uno de estos experimentos. Como resultado se obtiene una mayor comprensión de los aspectos más relevantes de cada experimento y se los puede comparar rápidamente según distintos aspectos relevantes.

Cuadro 3.9: Decisiones de Diseño

| Exp | Técnicas | Clasificación de faltas | Variables de respuesta |
|---|---|---|---|
| B-S | Testing funcional, testing estructural y revisión de código | Definida por Basili | Número y porcentaje de faltas detectadas, tiempo total de detección y tasa de detección de faltas. Para las técnicas de testing funcional y testing estructural: número de ejecuciones, tiempo de CPU consumido, máxima cobertura de sentencias lograda, tiempo de conexión utilizado, número y porcentaje de faltas observables a partir de los datos, y porcentaje de faltas observables a partir de los datos que efectivamente son observadas por los sujetos |
| K-L | Testing funcional, testing estructural y revisión de código | Definida por Basili. | Motivación en cada día. Habilidades con el lenguaje. Tiempo que han trabajado. Habilidades aplicando las técnicas de detección de defectos. Tiempo invertido en cada paso. Número de fallas reveladas. Número de fallas observadas. Número total de faltas detectadas. Número de faltas detectadas puramente por casualidad. Número de faltas detectadas utilizando las técnicas |
| M-M | Revisión de código | No clasifica | Por cada sujeto y cada grupo se registra el número de defectos detectados correctamente y el número de "falsos positivos". Las ganancias y pérdidas en las inspecciones grupales. La frecuencia de detección para cada defecto, tanto en las inspecciones basadas en papel como basadas en la herramienta |
| J-V(1) | Testing funcional, testing estructural y revisión de código | Sub clasificación de Basili (clases consideradas: inicialización, control y cosméticas) | Número de sujetos que detectaron una falta para cada falta en el programa |
| J-V(2) | Testing funcional, testing estructural y revisión de código | Idem experimento 1 | Número de sujetos que generan un caso de prueba capaz de detectar la falta |

Cuadro 3.10: Características del Proceso

| Exp | Organización de los sujetos | Capacitación | Proceso |
|---|---|---|---|
| B-S | 8 sujetos avanzados, 24 intermedio y 42 junior. Primera sesión (29 sujetos), segunda (13) y tercera (32) | La primer sesión de cada fase constituye la capacitación inicial. Los sujetos son expuestos a clases similares de capacitación | En cada fase se utilizan tres de los cuatro programas, y todos los sujetos utilizan las tres técnicas y verifican los tres programas. Cada fase se compone de 5 sesiones: una capacitación inicial, tres sesiones de pruebas y una sesión de seguimiento. Dentro de cada fase todos los sujetos verifican el mismo programa el mismo día |
| K-L | Primera réplica: 27 sujetos. Segunda réplica: En el primer día participaron 23 estudiantes, mientras que en el segundo 19 y en el tercero 15 | Antes de realizarse el experimento se presentan las técnicas de verificación a los sujetos y se ejercitan mediante una práctica | El experimento consiste en dos réplicas internas, las cuales son realizadas en tres días fijados en una semana. En cada día se verifica un programa distinto con las tres técnicas. Se utilizaron las mismas guías que en la práctica de ejercitación |
| M-M | La sesión 1 consiste en seis grupos de tres sujetos y un grupo de cuatro, mientras que la sesión 2 contiene siete grupos de tres sujetos | En las primeras seis semanas del experimento se proporciona formación a los alumnos | El proceso de inspección se realiza en 2 etapas. La primer etapa es de detección individual y la segunda etapa es de reunión grupal donde se realiza la consolidación. El experimento se realiza durante un período de diez semanas, en las últimas cuatro semanas se ejecutan las inspecciones. La inspección propiamente dicha consiste en utilizar la especificación del programa y la lista de funciones de librería para inspeccionar el código fuente, haciendo uso de una lista de verificación |
| J-V(1) | Se dividen en 8 grupos de 12 personas (de las cuales 4 de los grupos prueban con técnicas estructurales y 4 con técnicas funcionales) y 4 grupos de 25 (probando con revisión de código) | La sesión inicial es destinada al aprendizaje | El experimento fue organizado en 5 sesiones, en las últimas 4 se verifican los programas. Cada sesión se ejecuta en un día, para un programa, por 3 grupos que ejecutan cada uno una de técnica distinta cada uno de las 3 posibles |
| J-V(2) | Se dividen en 6 grupos de 7 a 8 integrantes | | El diseño de este experimento cambió, en este caso todos los grupos ejecutan todas las técnicas. El experimento se organiza en 3 días, en donde en cada día se ejecuta un programa y todas las técnicas. Cada técnica es ejecutada por dos grupos distintos |

Cuadro 3.11: Conclusiones

| Exp | Conclusión general | Conclusiones complementarias |
|---|---|---|
| B-S | Se observa que la técnica de revisión de código es tan efectiva como las técnicas de testing estructural y funcional en cuanto al número de faltas detectadas y costo asociado. La eficacia, eficiencia y costo dependen del tipo de software verificado | Los sujetos del nivel de experiencia avanzado detectaron más faltas y fueron más eficientes utilizando revisión de código que utilizando las técnicas de testing estructural y testing funcional, además detectaron más faltas con testing funcional que con estructural. Los sujetos pertenecientes a los niveles intermedio y junior fueron aproximadamente iguales de eficaces y eficientes con las tres técnicas. La técnica de revisión de código detecta más faltas de interfaz que las otras técnicas, mientras que testing funcional detecta más faltas de control. Los sujetos estimaron mejor el porcentaje de faltas detectadas cuando aplicaron revisión de código y peor cuando utilizaron testing funcional |
| K-L | Bajo las condiciones de sujetos relativamente inexperientes con el lenguaje y con las tres técnicas estudiadas, si no se considera al tiempo como un punto de interés, cualquiera de las técnicas puede ser utilizada con igual efectividad en la detección de defectos | Los sujetos fueron más eficientes al aplicar testing funcional |
| M-M | No hay diferencia significativa de eficiencia entre las inspecciones basadas en papel y las basadas en la herramienta, ya sea para los sujetos trabajando en forma individual como en equipo | Tampoco se presentan diferencias significativas entre ambos métodos en cuanto al número de falsos positivos reportados, y el costo ganado o perdido a causa de las reuniones grupales |
| J-V(1) | El número de personas que detecta una falta depende del programa que se está probando, la técnica utilizada y la falta. Existen faltas que se comportan mejor para ciertos programas y faltas que se detectan mejor con el uso de ciertas técnicas | Se observa también que la técnica de revisión de código es menos sensible a ocultar faltas que las otras técnicas. A pesar de que la técnica funcional se comporta mejor que la técnica estructural en la mayoría de los casos, los casos en que las dos técnicas que se comportan de forma idéntica o mejor que otra se producen indistintamente para cada tipo de falta |
| J-V(2) | La revisión de código siempre se comporta peor que las técnicas funcionales y estructurales, independientemente de la falta. En cuanto a la comparación entre las técnicas funcionales y estructurales, encontramos que las dos técnicas se comportan igual. La versión influye en el número de personas que detectan una falta | El número de sujetos que detecta un error utilizando la técnica de revisión no depende de la visibilidad de la falta. Independientemente del programa, la técnica y la falta, más sujetos generan casos de prueba que detectan las faltas en una versión y menos en otra versión |

# Capítulo 4

# Cubrimiento de Sentencias y Todos los Usos

En este capítulo se detallan las técnicas de verificación aplicadas en el experimento realizado. Se presentan dos técnicas de verificación, ambas de caja blanca, una basada en el flujo de control del programa y la otra basada en el flujo de datos. Dichas técnicas son **Cubrimiento de Sentencias** y **Todos los Usos** respectivamente.

En la sección 4.1 se explica la técnica Cubrimiento de Sentencias aplicando la misma en dos ejemplos. Al ser esta técnica ampliamente conocida no se profundiza en la misma. En la sección 4.2 se presenta *Data Flow Testing* donde se definen conceptos y establecen ciertos criterios necesarios para su aplicación. Incluye las subsecciones: 4.2.1 en la que se detalla cómo aplicar *Data Flow Testing* en algunas sentencias de código conocidas, 4.2.2 donde se presenta la aplicación de *Data Flow Testing* en orientación a objetos, y 4.2.3 que define la técnica de Todos los Usos. En esta última subsección se muestra la aplicación de la técnica mediante dos ejemplos.

## 4.1. Cubrimiento de Sentencias

Para cumplir con el criterio de Cubrimiento de Sentencias se debe asegurar que cada instrucción del código se ejecuta al menos una vez en el conjunto de casos de pruebas (CCP) construido.

Para entender mejor la técnica se aplica la misma en dos ejemplos sencillos. Se comienza aplicando la técnica al siguiente segmento de código:

```
public int func(int a, int b){
  x = a + b;
  if (a > 0){
      y = 2;
  }else{
      x = x + 2;
  }
  b = x + a;
  return b;
```

```
}
```

Como punto de partida se realiza el Grafo de Flujo de Control de forma de poder visualizar más fácilmente las secuencias de sentencias que se deben ejecutar para cumplir el criterio de cubrimiento. La figura 4.1 ilustra el diagrama.

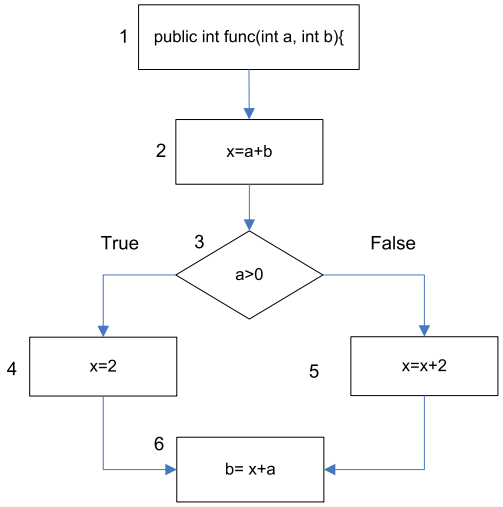



Figura 4.1: Grafo de Flujo de Control - Ejemplo 1

Para cumplir con el criterio debemos ejecutar al menos una vez cada instrucción del código. Se observa en el diagrama que no es posible encontrar una única secuencia de instrucciones que cumpla el criterio. La secuencia **1-2-3-4-6** ejecuta las instrucciones de código correspondientes a la evaluación *True* del **if**. Sin embargo, con esta secuencia no se está ejecutando la sentencia del nodo 5. Es necesario agregar otra secuencia para cumplir el criterio. La secuencia **1-2-3-5-6** ejecuta la sentencia que faltaba, correspondiente a la evaluación *False* del **if**.

Se observa con este ejemplo que no siempre es posible encontrar un único caso de prueba que ejecute todas las instrucciones del código. En este caso se cumple con el criterio de Cubrimiento de Sentencias encontrando dos casos de prueba que logran ejecutar todas las instrucciones.

La secuencia **1-2-3-4-6** es ejecutable con un caso de prueba (CP) que asigna a la variable **a** un valor mayor a cero. Se determina por ejemplo **a**=3. La secuencia **1-2-3-5-6** es ejecutable con un CP que asigna por el contrario un valor menor o igual a cero a la variable **a**. Para este caso se determina por ejemplo **a**=-1. En ambos casos el valor de **b** puede ser cualquiera, se elige el valor 1 para el ejemplo. Se concluye este ejemplo determinando el conjunto de casos de pruebas **(a=3, b=1);(a=-1,b=1)** que cumple con el criterio. Para completar los casos de prueba deben especificarse los resultados esperados. El primer CP queda determinado por los valores de entrada (**a**=3,**b**=1) y el resultado esperado **b**= 9, mientras que para el segundo CP se determinan (**a**=-1,**b**=1) y **b**= 1.

En el segundo ejemplo se aplica la técnica al método ordenarSinRep de la clase OrdenadorSinRep, perteneciente al programa que es utilizado durante la

etapa de entrenamiento de los sujetos en el experimento realizado. El programa se presenta en el capítulo 6 correspondiente a las Experiencias Iniciales. La figura 4.2 ilustra el Grafo de Flujo de Control del método.

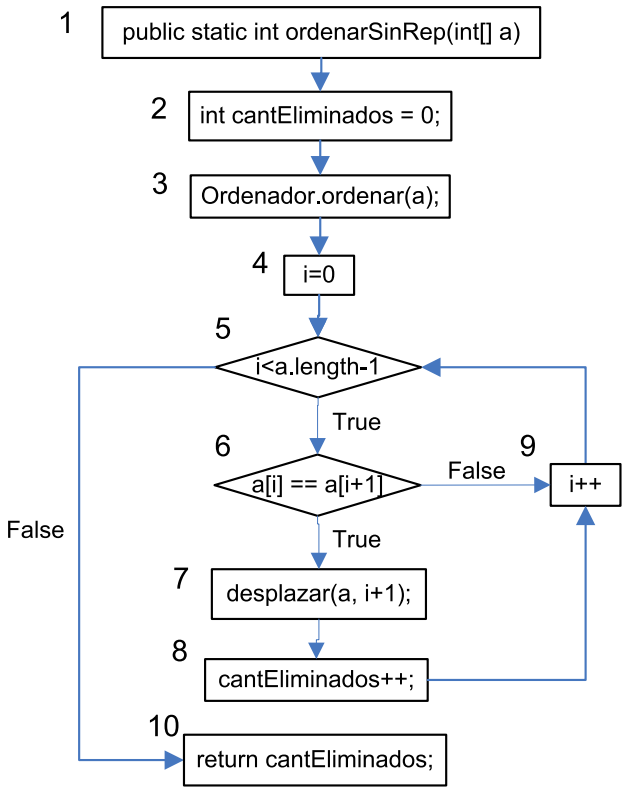



Figura 4.2: Grafo de Flujo de Control para ordenarSinRep

Para asegurar que se ejecute al menos una vez cada sentencia del código se puede ejecutar la secuencia **1-2-3-4-5-6-7-8-9-5-10**. Un posible valor a asignar al array de entrada que provoca la ejecución del camino identificado es a=[2,2]. Se determina para este ejemplo el conjunto de casos de prueba **a=[2,2]** que cumple con el criterio. La figura 4.3 muestra la secuencia elegida que asegura el cumplimiento de la técnica.

Se considera que la técnica Cubrimiento de Sentencias es lo suficientemente sencilla y conocida, por lo que no se realiza un análisis más exhaustivo de la misma.

## 4.2. Data Flow Testing

Data Flow Testing (DFT) se define como un conjunto de técnicas de prueba que observa cómo los distintos valores asociados a variables pueden afectar la ejecución de un programa. DFT no sólo explora el flujo de control de los programas, sino que también presta atención a cómo se define una variable y cómo se utiliza a lo largo del flujo de control. Como se mencionó en la introducción,

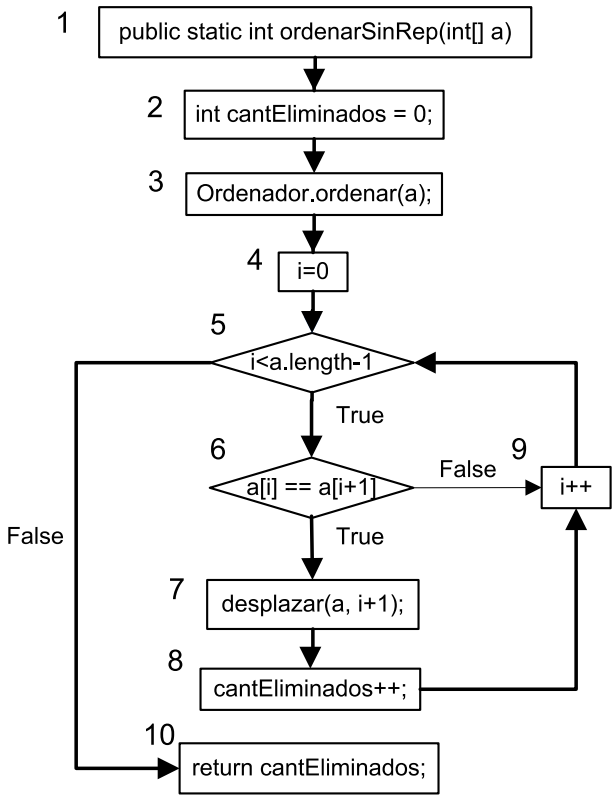



Figura 4.3: Secuencia de ejecución para ordenarSinRep

las técnicas basadas en el flujo de datos del programa expresan los cubrimientos del testing en términos de las asociaciones definición-uso del programa.

Algunos términos necesarios para poder comprender mejor DFT se detallan a continuación:

- Una **definición** de una variable ocurre cuando se asigna algún valor a la misma, ya sea de forma explícita o implícita.
- Una variable en el lado derecho de una asignación es llamada un **uso computacional** o un **c-uso**. Existen casos particulares que serán presentados en detalle más adelante.
- Una variable en un predicado booleano que implique una bifurcación en el código resulta en un par **uso predicado** o **p-uso** correspondiendo a las evaluaciones verdadero y falso del predicado. Existen casos particulares que serán presentados en detalle más adelante.
- Un camino (i,$n_1$,$n_2$,...,$n_m$,j) es un camino **limpio-definición** respecto a la variable x desde la salida de i hasta la entrada a j si no contiene una definición de x en los nodos de $n_1$ a $n_m$.

Las definiciones pueden realizarse de forma explícita o implícita. Un ejemplo de definición explícita es **a=3** donde se ve claramente una asignación mediante

el signo de igualdad. Como ejemplo de definición implícita se cita la sentencia **a++**. En dicha sentencia no se ve un signo de igualdad, sin embargo, se sabe que la sentencia puede traducirse a **a=a+1** donde sí se observa claramente que se está asignando un valor a la variable **a**. Además, se identifica un c-uso de la variable **a** ya que se encuentra en el lado derecho de una asignación.

En la mayoría de la literatura que presenta técnicas basadas en el flujo de datos se brindan ejemplos que contienen variables simples como enteros o booleanos. Sin embargo, al momento de aplicar estas técnicas sobre variables complejas como arrays u objetos se deben definir ciertos criterios, que normalmente no son establecidos. Especificar estos criterios es fundamental para definir correctamente la técnica a aplicar y permitir conocer bajo qué condiciones se aplicó la misma. Distintos criterios pueden ocasionar diferentes resultados en la efectividad y el costo de DFT. A continuación se presentan los criterios más relevantes definidos para la aplicación de la técnica durante el experimento.

Se realiza un estudio de varios artículos para conocer los criterios establecidos en cada uno. Dentro de los artículos inspeccionados se encuentran, "An Applicable Family of Data Flow Testing Criteria" [5], "Interprocedural data flow testing" [8] y "Performing Data Flow Testing on Classes" [7].

Se debe determinar un criterio que establezca cómo deben considerarse los arrays en DFT. En el artículo "An Applicable Family of Data Flow Testing Criteria" [5] se establece que una asignación de **a[e]** siendo **a** un array consiste en un c-uso de cada variable que aparece en la expresión **e** y una definición de la variable **a**. Por otro lado, una ocurrencia de **a[e]** que no sea una asignación consiste en un uso de cada variable que aparece en la expresión **e** y un uso de la variable **a**.

Consideramos que no es viable distinguir las definiciones y usos a nivel de las posiciones del array. El motivo es la imposibilidad de determinar a qué posición se corresponde una expresión cuando es utilizada como índice del array. Una situación en la se puede ver esta dificultad es cuando se presenta en el código una ocurrencia de un array en la que el índice está determinado de forma explícita (por ej.: a[3]), y otra ocurrencia para el mismo array donde el índice depende de la evaluación de una expresión (por ej.: a[e]). Se observa que en la primer ocurrencia se está accediendo a la posición 3 del array, pero en la segunda ocurrencia el valor que toma **e** depende del flujo de datos de cada ejecución, y no puede determinarse independientemente de los valores de entrada que se asignen. Carece de sentido considerar la definición y/o uso de la posición 3 del array en la primer ocurrencia dado que en la segunda ocurrencia se debe considerar la definición y/o uso del array como estructura atómica.

En la figura 4.4 se presenta un ejemplo sencillo para ilustrar la situación descrita anteriormente.

Solamente se identifican las definiciones y usos de la variable **a**. En el nodo 1 se realiza una definición, en el nodo 2 se realiza un p-uso y en el nodo 5 un c-uso de **a**. Si se distinguen las posiciones del array, en el nodo 1 se tiene una definición de la posición 3 de **a** y en el nodo 2 se identifica un p-uso de la posición 3 de **a**. Sin embargo, en el nodo 5 no es posible determinar a que posición del array corresponde el c-uso debido a que no es posible conocer el valor de **i**. Si la expresión booleana del nodo 2 evalúa *True*, **i** toma el valor 2 en el nodo 5 lo que causa un c-uso de la posición 3 de **a**. En cambio si la expresión evalúa *False* la posición del array accedida en el nodo 5 queda sujeta al valor que tome la variable **i** en el nodo 4. De esta manera se observa que el c-uso del nodo 5

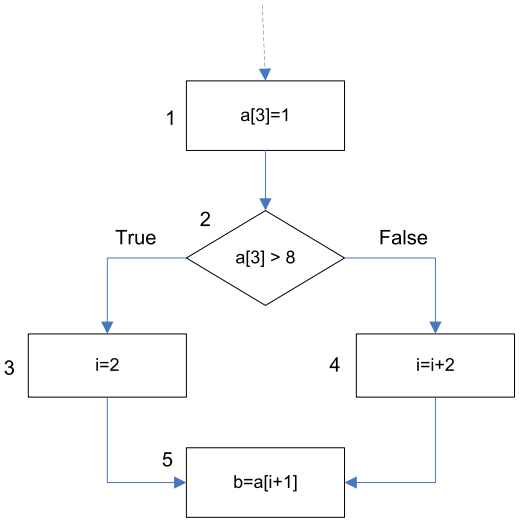



Figura 4.4: Grafo de Flujo de Control - Ejemplo Array

corresponde a diferentes posiciones del array **a** dependiendo del camino que se tome en el CFG. Por lo tanto consideramos que las definiciones y usos son a nivel de la variable **a** y no de sus posiciones.

También es de importancia establecer un criterio para identificar las definiciones y usos de punteros y referencias a objetos. En el artículo "An Applicable Family of Data Flow Testing Criteria" [5] se establecen los criterios necesarios para identificar definiciones y usos en punteros. El autor explica que ignora a los punteros dado que no es posible determinar estáticamente la locación de memoria de los mismos. Con la utilización de alias el uso de un mismo sector de memoria puede variar dependiendo del camino del grafo de flujo de control que se recorre, debido al empleo de variables de distinto nombre para referenciar a la misma locación de memoria.

El criterio que adoptamos es no tener en cuenta la utilización de alias debido a las dificultades planteadas anteriormente, con lo cual sólo consideramos que se corresponden las definiciones y usos de variables del mismo nombre.

La figura 4.5 ilustra un ejemplo sencillo para entender el motivo de establecer el criterio de ignorar el uso de alias.

En este ejemplo se consideran dos clases llamadas Persona y Empresa. Se define la variable **p** de tipo Persona, y las variables **e** y **e1** de tipo Empresa. El objeto **p** tiene un pseudo-atributo **emp** de tipo Empresa. No se identifican las definiciones y usos de **p** dado que no influyen en el problema que se quiere mostrar. Para desarrollar el ejemplo se considera el uso de alias de forma de ilustrar las dificultades que ello implica. En el nodo 1 se realiza una definición de la variable **e**, convirtiéndola en un alias del pseudo-atributo de **p**. En el nodo 3 existe un c-uso del pseudo-atributo de **p**.

Si el flujo de ejecución del programa toma la rama de la izquierda, donde se ejecutan sentencias que no afectan dichas variables, sucede que el pseudo-atributo de **p** coincide con **e** al alcanzar el nodo 3. Por lo cual, el c-uso de dicho nodo corresponde tanto al pseudo-atributo de **p** como a **e** dado que son alias.

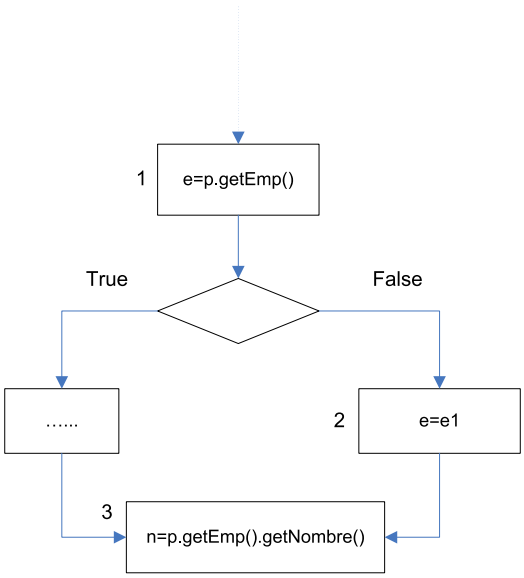



Figura 4.5: Grafo de Flujo de Control - Ejemplo Puntero

Sin embargo, si el flujo de ejecución del programa toma la rama de la derecha, en el nodo 2 se realiza una definición de **e** haciendo que apunte a la locación de memoria de la empresa **e1**. Al alcanzar el nodo 3 por dicha rama la variable **e** ha dejado de ser un alias del pseudo-atributo de **p**, por lo tanto el c-uso efectuado en dicho nodo corresponde sólo al pseudo-atributo de **p**.

Se observa con este ejemplo que los c-usos/p-usos asociados a las variables dependen del camino del CFG que se recorra, imposibilitando la determinación de los mismos de forma independiente de los valores de entrada que se elijan. Estas dificultades presentadas son las que nos llevan a establecer como criterio no tener en cuenta la utilización de alias, con lo cual sólo consideramos que se corresponden las definiciones y usos de variables del mismo nombre. Entonces, si identificamos las definiciones y usos de las variables que aparecen en el CFG se tienen: una definición de **e** y un c-uso de **p** en el nodo 1, una definición de **e** y un c-uso de **e1** en el nodo 2, y una definición de **n** y un c-uso de **p** en el nodo 3.

### 4.2.1. Clasificación de Sentencias

En esta sección se indica cómo identificar las definiciones y usos en diferentes sentencias de código utilizadas frecuentemente. Se detallan a continuación los criterios establecidos en el artículo "An Applicable Family of Data Flow Testing Criteria" [5]:

- v = expresión
  Se considera un c-uso para cada variable en expresión y una definición de v

- read($v_1$, $v_2$, . . . $v_n$)
  Se considera una definición de cada $v_1$ . . . $v_n$

- write($v_1$, $v_2$, . . . $v_n$)
  Se considera un c-uso de cada $v_1$ . . . $v_n$

- while B do S
  Se considera un p-uso por cada variable en la expresión booleana B, S se clasifica dependiendo de su composición.

- for v:=el to e2 do S
  Se considera una definición de v, un c-uso por cada variable en e1 y un p-uso por cada variable en e2. S se clasifica dependiendo de su composición.

- if B then $S_1$ else $S_2$
  Se considera un p-uso por cada variable en la expresión B, $S_1$ y $S_2$ se clasifican dependiendo de su composición.

- case e1 $S_1$: $S_2$:
  Se considera un p-uso por cada variable en la expresión e1, $S_1$ y $S_2$ se clasifican dependiendo de su composición.

Se establecen además criterios para sentencias no explicados en los artículos estudiados pero que son de gran relevancia para el experimento.

- v++
  Es una sentencia típica de Java. Se puede traducir a una sentencia v=v+1, por lo tanto, se considera una definición y un c-uso de la variable v.

- return v
  Se considera un c-uso de la variable v.

- return v >0
  Se considera un c-uso de la variable v. No es un p-uso porque, aunque v está en un predicado booleano, este no resulta en una bifurcación en el código. Aplica para cualquier expresión booleana que contenga a v y no implique una bifurcación.

- for (v=e1, v<= e2, v++)
  Esta sentencia se traduce en tres sentencias como se ilustra en la figura 4.6: la inicialización v=e1, la condición de parada v<= e2, y el incremento v++. Estas sentencias pertenecen a nodos distintos del CFG de la sentencia for. En la inicialización se considera una definición de v y un c-uso por cada variable en e1. En la condición de parada se considera un p-uso de v y un p-uso por cada variable en e2. En el incremento se considera una definición y un c-uso de v.

### 4.2.2. Data Flow Testing en Orientación a Objetos

En Orientación a Objetos la unidad básica de testing es la clase. Es necesario probar sus métodos de forma individual y de forma colectiva, de manera de verificar las interacciones que se generan a través de las secuencias de llamadas provocadas por la invocación de un método en particular. Existe un tercer nivel de interacción que se origina como resultado de la ejecución de secuencias arbitrarias de métodos accesibles desde fuera de la clase (métodos públicos).

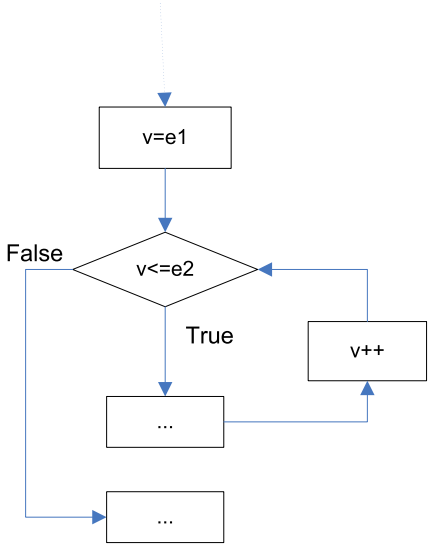



Figura 4.6: Grafo de Flujo de Control que ilustra como se traduce una sentencia de código FOR

Los conceptos que se presentan a continuación son abarcados en el artículo "Performing Data Flow Testing on Classes" [7]. Las técnicas de DFT pueden ser aplicadas tanto para el testing de métodos individuales pertenecientes a una clase, como para métodos que interactúan con otros métodos de la misma o de otras clases. Dicha interacción puede deberse a la cadena de llamadas provocada por la invocación de un método en particular, como a la ejecución de secuencias arbitrarias de métodos por parte de un usuario de la clase. El testing de una clase puede realizarse en tres niveles diferentes, los cuales se definen a continuación:

**Intra-método**: Para cumplir con la técnica se considera solamente el método bajo prueba. Entonces, al momento de diseñar los casos de prueba no se considera el código de los métodos que interactúan con el método bajo prueba para la identificación de definiciones y usos de variables (excepto para atributos como se explica más adelante en el capítulo).

**Inter-método**: Para cumplir con la técnica se considera el código de los métodos que interactúan con el método bajo prueba para la identificación de definiciones y usos de variables. Corresponde al testing de métodos públicos junto con todos los métodos que son alcanzables desde él (es decir, que pertenecen a la secuencia de llamadas que dicho método dispara).

**Intra-clase**: Corresponde al testing de las interacciones de los métodos públicos de una clase cuando son invocados en secuencias arbitrarias. Como existe un número infinito de combinaciones solo es posible verificar un subconjunto de las mismas.

En relación a los niveles presentados anteriormente se identifican tres tipos de pares definición-uso a ser verificados en una clase.

**Pares Intra-método**: Son los que tienen lugar en métodos individuales y verifican el flujo de datos limitado a dichos métodos. Tanto la definición como el uso deben pertenecer al método bajo prueba.

**Pares Inter-método**: Ocurren cuando interactúan métodos en el contexto de la invocación de un único método público. Son pares donde la definición pertenece a un método y el correspondiente uso se sitúa en un método llamado o en un método llamador (tanto llamadas directas como indirectas) en la misma cadena de invocaciones. El alcance del par definición-uso sobrepasa la frontera de algún método.

**Pares Intra-clase**: Tienen lugar en el contexto de invocaciones a secuencias de métodos públicos. El alcance del par definición-uso sobrepasa la frontera de al menos dos métodos públicos de la clase.

El experimento realizado sólo abarca el testing a nivel de Intra-método e Inter-método, por lo que Intra-clase no se explica en detalle dado que se encuentra fuera de nuestro alcance. El motivo de esta decisión es simplificar la realización del experimento.

### Pruebas a nivel de Intra-método

En las pruebas a nivel de Intra-método solamente se consideran los Pares Intra-método. Esto quiere decir que las definiciones y usos considerados son solamente los que tienen lugar dentro del método bajo prueba. Sin embargo, en el código del método bajo prueba pueden existir llamadas a otros métodos, y éstas deben ser tenidas en cuenta de alguna forma. Dichas llamadas normalmente implican un intercambio de parámetros. Entonces, hay que definir un criterio que establezca cómo considerar las variables que intervienen en invocaciones a otros métodos en lo que refiere a la identificación de definiciones y usos.

En el artículo "Interprocedural data flow testing" [8] se presenta un ejemplo para ilustrar algunos problemas involucrados en el testing a nivel de Intra-método. En dicho ejemplo al probar un método de forma individual, para las variables que intervienen en llamadas a otros métodos, se consideran definiciones o usos según la participación de las mismas en el método invocado. Si la variable es utilizada en el método invocado se considera un uso, y si la variable es asignada dentro del mismo se considera una definición.

En el artículo "An Applicable Family of Data Flow Testing Criteria" [5] para las variables que intervienen en llamadas a otros métodos se consideran siempre usos. Si una variable es pasada por referencia se considera además una definición de la misma.

Optamos por establecer un criterio en base a los dos anteriores. Definimos que, cuando la variable es sólo de entrada al método se considera un uso, en caso de ser sólo de salida se considera una definición, y en caso de ser de entrada-salida se considera una definición y un uso. El uso se clasifica en c-uso o p-uso dependiendo de la sentencia de código en la que se ubique la invocación al método. Si la invocación se encuentra en un predicado booleano que implica una bifurcación se considera un p-uso, en caso contrario se considera un c-uso.

Dado que los programas utilizados en el experimento se encuentran codificados en Java se deben tener en cuenta las particularidades de dicho lenguaje y realizar consideraciones al respecto en los criterios definidos para la aplicación de DFT. En Java los parámetros son pasados por valor, por lo que todos los parámetros (excepto los arrays) en las llamadas a otros métodos son únicamente de entrada. Por lo tanto, no van a existir definiciones de variables en las invocaciones a otros métodos. Los arrays constituyen una excepción a esta regla. Como ya mencionamos consideramos que las definiciones y usos de los array son a ni-

vel del array (en su totalidad) y no a nivel de sus posiciones. Esto significa que una asignación a alguna posición de un array implica una definición del mismo. Dado que las modificaciones que se realizan a un array en el método invocado son visibles desde el método que lo invoca (método bajo prueba), consideramos que las variables de tipo array sí pueden ser variables de salida en Java, y por lo tanto pueden ser definidas en la invocación a otro método.

También se debe establecer cómo considerar a los atributos pertenecientes a la clase del método bajo prueba. Definimos que al verificar un método que invoca a un método de la misma clase, se deben tener en cuenta las definiciones y usos de los atributos de la clase en el método invocado. Se asocia una definición y/o uso del atributo en el nodo del CFG correspondiente a la invocación. El uso se clasifica en c-uso o p-uso según si la invocación se encuentra en un predicado booleano que implique bifurcación o no. Además, en el nodo inicial del CFG de cada método a probar se consideran definiciones de todos los atributos de la clase que son utilizados en el método. Estos criterios no se encuentran ejemplificados en el informe para no incorporar complejidad a la lectura.

Se utiliza el método ordenarSinRep de la clase OrdenadorSinRep (al igual que en Cubrimiento de Sentencias) para presentar un ejemplo de aplicación de Intra-método. La figura 4.2 ilustra el GFC para el método ordenarSinRep. Para simplificar el ejemplo no se identifican las definiciones y usos en todos los nodos del grafo, sino que se analizan sólo los nodos que son de mayor interés para mostrar la identificación de definiciones y usos para el nivel de Intra-método. En el nodo 1 se identifica una definición de la variable **a** de tipo array por tratarse de un parámetro del método bajo prueba. En el nodo 3 se invoca a otro método pasando la variable **a** por parámetro. Revisando el código del método invocado se clasifica a la variable como parámetro de entrada-salida. Esto se debe a que la variable es utilizada y definida (variable de tipo array) en el método invocado. Por lo tanto, se identifica una definición y un uso de la variable **a** en el nodo 3. Como la invocación al método no se encuentra en un predicado booleano que implique una bifurcación el uso corresponde a un c-uso. Se observa una situación similar en el nodo 7, donde se invoca a otro método pasando las variables **a** e **i (i+1)** por parámetro. El razonamiento para la variable **a** es análogo que para el nodo 3. La variable se clasifica como parámetro de entrada-salida, por lo cual se identifica una definición y un uso. La variable **i** se clasifica como parámetro de entrada al método, identificando entonces un uso de la misma. Al igual que para el nodo 3, los usos de las variables **a** e **i** en el nodo 7 corresponden a c-usos.

### Pruebas a nivel de Inter-método

Para las pruebas a nivel de Inter-método se deben considerar las variables que aparecen en el método bajo prueba y que son pasadas por parámetro a otros métodos. También se consideran los atributos de la clase que sean definidos o usados en el método a probar y que sean definidos o usados en un método invocado (el método invocado necesariamente debe ser de la misma clase en este caso). Se debe contar con la información de las definiciones y usos de dichas variables (y atributos) que ocurren en los métodos que pertenecen a la secuencia de llamadas disparada por el método bajo prueba. Las variables pueden ser definidas en el método a probar y usadas en algún método invocado, así como pueden ser usadas en el método a probar siendo definidas en algún método invocado. Se consideran los métodos pertenecientes a la secuencia de llamadas provocada por

el método bajo prueba, sean invocados de forma directa o indirecta. Se define invocación de forma directa cuando la llamada es realizada de forma explícita por el método verificado, e indirecta cuando la llamada es realizada a lo largo de la secuencia de llamadas iniciada por el método verificado.

En Java los parámetros **x** recibidos por un método invocado por el método bajo prueba se consideran de la siguiente forma en el método invocado:

- Se consideran todos los usos de la variable **x** previos a la primer definición de **x**.
- No se consideran las definiciones de **x**.
- En caso de tener usos de **x** posteriores a una posible definición de **x** (p.e: la definición está condicionada a un IF) consideramos dichos usos de **x**.

La consideración anterior es válida para todo parámetro **x** que no sea de tipo array. Si el método bajo prueba invoca otro método con un parámetro **x** de tipo array se consideran todas las definiciones de **x** y todos los usos de **x** en el método invocado. Esto se debe a que las variables de tipo array pueden ser parámetros de salida en Java, como fue explicado en 4.2.2 .

En el artículo "Performing Data Flow Testing on Classes" [7] se presenta un enfoque que da soporte a la aplicación de DFT para todos los tipos de interacción de flujo de datos en una clase. Con el propósito de registrar la información necesaria se desarrolla una nueva representación gráfica de la clase, denominada Grafo de Flujo de Control de la Clase (CCFG). Como el CCFG abarca la aplicación de Intra-clase y no es de nuestro interés el estudio de dicho nivel, definimos una variante que denominamos CFG', la cual abarca solo los niveles de Intra-método e Inter-método.

Las principales diferencias de CFG' con CCFG son que: se realiza un CFG' para cada método bajo prueba mientras que el CCFG se realiza para toda la clase, esto se debe a que el CCFG representa el nivel de Intra-clase y el CFG' no lo hace. Además, el grafo CFG' incluye a los métodos de otras clases que son invocados por el método bajo prueba (no se limita a la clase del método bajo prueba).

Se realiza a continuación una descripción de los pasos a seguir para la construcción del CFG':

```
Entradas: C (la clase a testear), y las clases accedidas
desde la misma
Salida: G (el CFG')
Inicio
    // Paso 1: Construir el grafo de llamadas
    G = Grafo de llamadas

    // Paso 2: Reemplazar cada nodo de llamada por el
correspondiente grafo de flujo de control
    Para todo método M en G hacer
        Reemplazar el nodo de llamada de M en G por el CFG de M
        Actualizar las aristas de forma apropiada
    Fin Para Todo
```

```
    //Paso 3: Reemplazar las llamadas a métodos por nodos de
    call y return
    Para todo nodo de llamada S en G, que represente la
    invocación a un método M hacer
        Reemplazar S por nodos de call y return
        Actualizar las aristas de forma apropiada
    Fin Para Todo

    // Paso 4: Retornar el CFG'
    Retornar G
Fin
```

Percibimos la necesidad de establecer un criterio de parada que determine el nivel de profundidad del grafo, es decir, poder determinar cuando un método debe ser extendido o no (reemplazando la llamada al mismo por su CFG como se indica en el paso 2 del algoritmo). El motivo de establecer un criterio de parada se debe a que el grafo obtenido puede alcanzar un tamaño que dificulte notoriamente el testing, incluso podría generarse un grafo infinito. Un posible criterio es detener la extensión del grafo en un nivel determinado (por ej. nivel 3 de profundidad), dicho nivel puede especificarse dependiendo de la complejidad del grafo resultante. Otro posible criterio es detener la extensión cuando el CFG del método actual ya forma parte del CFG' en construcción. Podemos distinguir dos alternativas del criterio anterior, cuando el CFG se ubica en la misma rama que el método a extender, o cuando se ubica en un nivel anterior sin importar la rama. Consideramos nivel de profundidad 1 cuando estamos posicionados en un método llamado directamente por el método bajo prueba. Generalizando la definición anterior se considera que un método pertenece al nivel n+1 cuando es invocado directamente por un método del nivel n. El criterio que elegimos es detener la extensión del grafo en el nivel 2 de profundidad.

Se aplica ahora Inter-método al método ordenarSinRep de igual forma que se hizo para Intra-método. Se reemplazan las llamadas a los métodos **ordenar** y **desplazar** por su correspondiente grafo de flujo de control. En la figura 4.7 se ilustra el CFG' correspondiente. Se puede ver en la figura que se reemplaza la llamada al método **ordenar** por su GFC y se incluyen los nodos de *call* y *return*, lo mismo se hizo para el método **desplazar**. La variable involucrada en las llamadas a otros métodos por el método bajo prueba es sólo **a** de tipo array. La variable **i** no es considerada, dado que en el método **desplazar** no hay un uso de **i**. El uso es del valor **i+1** pasado por parámetro. Observando el GFC' se identifican las definiciones y usos de **a**: definición en el nodo 1, c-uso en el nodo 3, p-uso en el nodo 9, c-uso en el nodo 10, definición y c-uso en el nodo 11, definición en el nodo 12, p-uso en el nodo 18, p-uso en el nodo 19, p-uso en el nodo 21, y definición y c-uso en el nodo 22. De esta forma se obtienen las definiciones y usos aplicando Inter-método, esto es, observando las definiciones y usos de las variables involucradas en las invocaciones a otros métodos.

### Consideraciones para Intra-método e Inter-método

En esta sección se indican algunos criterios que aplican tanto para el testing a nivel de Intra-método como a nivel de Inter-método. Estas consideraciones no son presentadas en los artículos estudiados, sin embargo, son de gran aporte

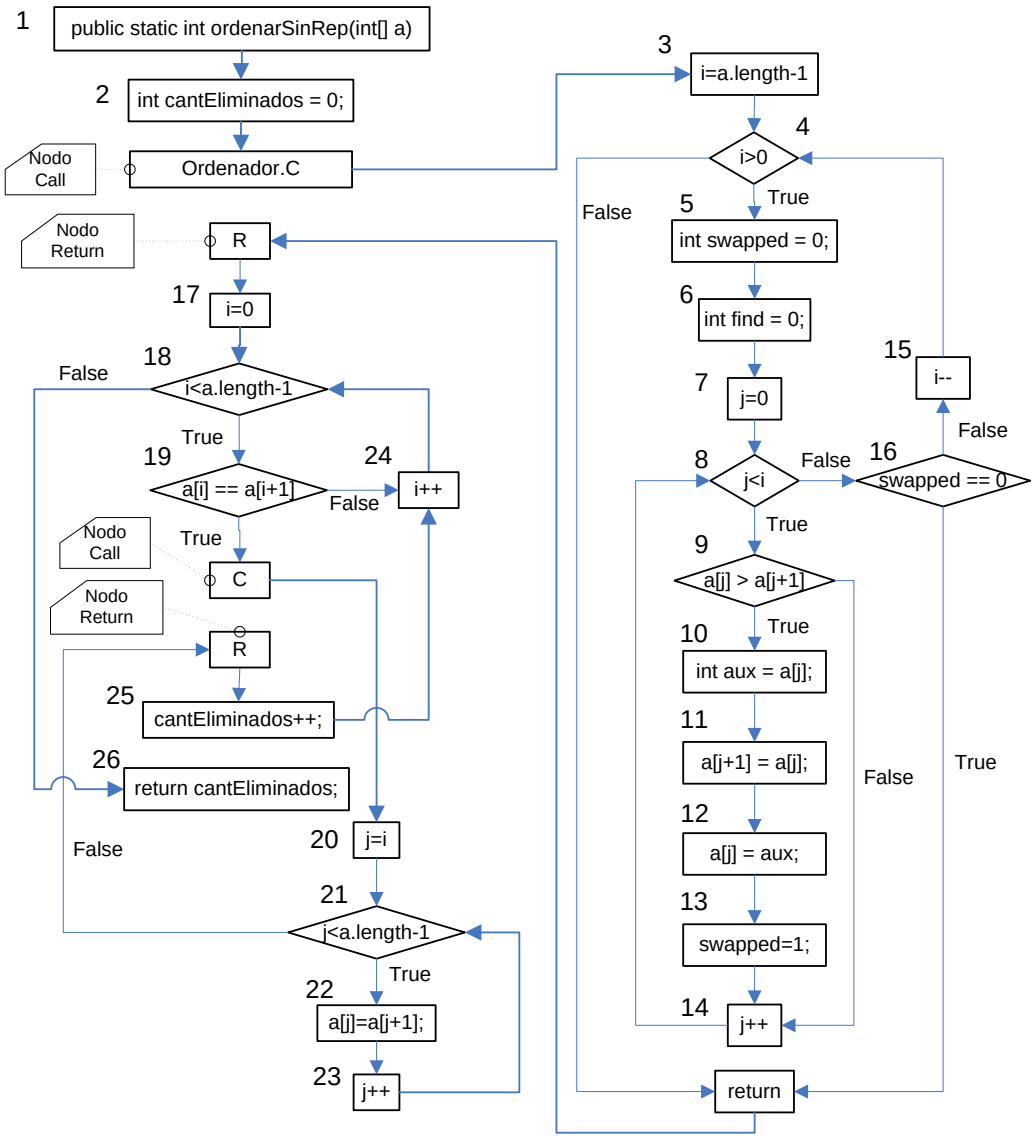



Figura 4.7: Grafo de Flujo de Control extendido (CFG') para el método ordenarSinRep

para lograr definir la técnica correctamente.

- **Parámetros recibidos por el método bajo prueba:** Se considera que en la firma del método bajo prueba se realizan las definiciones de todos los parámetros de entrada.
- **Variables locales en un método invocado por el método bajo prueba:** Las definiciones y usos de variables locales de un método invocado por el método bajo prueba no son considerados. El alcance de estas variables está determinado por el método invocado, por lo tanto no interactúan con el método bajo prueba.
- **Atributos en un método de otra clase invocado por el método bajo prueba:** No se consideran las definiciones y usos de atributos en un método de otra clase invocado por el método bajo prueba.
- **Try - Catch:** Al probar un método a que contiene try-catch sólo consideramos el código contenido en el bloque try.

### 4.2.3. Todos los Usos

En las secciones anteriores se describieron los criterios a tener en cuenta en DFT para el testing a niveles de Intra-método e Inter-método. Sin embargo, no se ha explicado aún la técnica Todos los Usos (TU) que es la que se aplica en el experimento. La técnica elegida es **Todos los Usos**. El cumplimiento de esta técnica exige que para cada par definición-uso (para cada variable) se ejecute **al menos un** camino libre-definición. Para los p-usos se debe llegar tanto con *True* como con *False*, es decir, el predicado booleano que contiene a la variable debe hacerse al menos una vez *True* y al menos una vez *False*.

Un camino libre-definición está asociado a una variable, es decir, se determina un camino que no contenga definiciones para dicha variable. Por lo tanto, al determinar un camino libre-definición para una variable x es posible que dicho camino contenga definiciones para otras variables.

Es válido que un camino libre-definición empiece y termine en un mismo nodo. Un caso típico de esto es cuando en un nodo se identifica una definición y un uso de la misma variable (p.e i++). Para encontrar un camino libre-definición para esa variable (para dicho par definición-uso) es necesario encontrar un bucle que comience en la definición y termine en el uso (esto es, que comience y termine en el mismo nodo).

Cabe resaltar que la técnica no exige nada respecto a la cantidad de caminos libre-definición seleccionados. La condición dice que para cada par definición-uso (para cada variable) se ejecute al menos un camino libre-definición. Un camino puede abarcar uno o varios pares definición-uso. También es válido que más de un camino llegue a un mismo uso.

Para el experimento se define TU como la aplicación de TU a nivel de Intra-método e Inter-método. Además, primero debe aplicarse Intra-método y luego Inter-método.

A continuación se realizan dos ejemplos aplicando TU. En el primer ejemplo se aplica sólo Intra-método por tratarse de un segmento de código sencillo sin interacción con otros métodos. En el segundo ejemplo, más realista, se aplica Intra-método e Inter-método.

La figura 4.8 ilustra el CFG del segmento de código que se utiliza para aplicar la técnica. En los nodos 2 y 5 se realizan definiciones de **x**, en el nodo 5 se realiza también un c-uso de **x**, y en el nodo 6 se realiza un p-uso de **x**. En los nodos 3 y 4 no existen ocurrencias de la variable **x**. Para cumplir con la técnica de Todos los Usos se debe ejecutar para cada par definición-uso al menos un camino libre-definición. Se identifican entonces los caminos libre-definición necesarios. Partiendo de la definición de **x** en el nodo 2 se encuentran los caminos libre-definición (2,3,5) y (2,3,4,6). El camino (2,3,5) que llega al c-uso del nodo 5 y (2,3,4,6) que llega al p-uso del nodo 6. Partiendo del otro nodo donde se define **x** (nodo 5) se encuentra el camino libre-definición: (5,6) que llega al p-uso del nodo 6. Partiendo del nodo 5 no es posible encontrar un camino libre-definición que llegue al c-uso del propio nodo 5. La inexistencia de dicho camino no impide cumplir con la técnica.

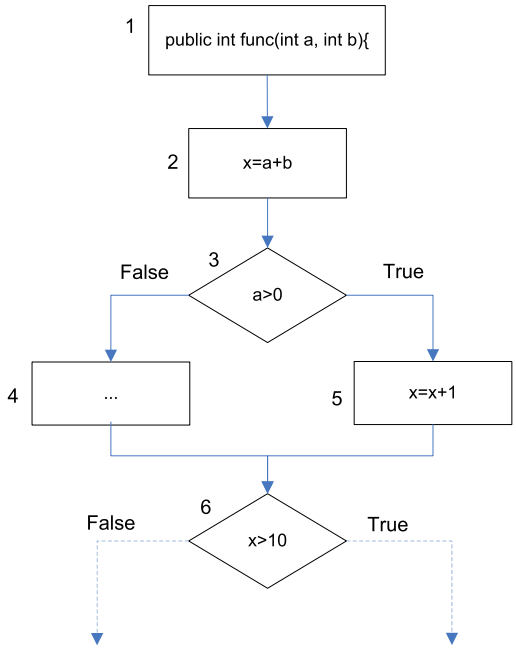



Figura 4.8: Grafo de Flujo de Control de un segmento de código

Se busca ahora un conjunto de casos de prueba tal que se ejecute, para las definiciones de **x**, al menos un camino libre-definición que lleve a cada c-uso y p-uso, es decir, que ejecute el conjunto de caminos que se ha identificado. Con los valores de entrada a=1 y b=3 se ejecuta el camino (2,3,5), y el camino (2,3,5,6) haciendo que la condición del nodo 6 evalúe en *False*. Para ejecutar el mismo camino haciendo que la condición del nodo 6 evalúe en *True* se asignan los valores de entrada a=8 y b=3. Se puede observar que con los mismos casos de prueba (CP) definidos anteriormente se ejecuta el camino (5,6) haciendo que la condición del nodo 6 evalúe en *True* y en *False*, con lo cual se encuentra un conjunto de casos de pruebas que cumple con el criterio. Es importante señalar que para este ejemplo solo se han establecido los valores de entrada, sin embargo, para que los casos de prueba estén completos se deben especificar también los

resultados esperados. Además se realiza el ejemplo sólo para la variable **x**, sin embargo, las variables **a** y **b** también deben ser tenidas en cuenta al aplicar la técnica, encontrando sus definiciones y usos.

Conjunto de caso de prueba: (a=1,b=3),(a=8,b=3).

Para el segundo ejemplo se opta por aplicar la técnica al método ordenarSinRep ya presentado. En las secciones anteriores se presentó cómo identificar las definiciones y usos para Intra-método e Inter-método sobre dicha operación. A continuación se realiza paso a paso la aplicación de la técnica Todos los Usos, primero a nivel de Intra-método y luego a nivel de Inter-método.

Para Intra-método el primer paso es identificar las variables que participan en el método **ordenarSinRep** junto con sus definiciones y usos. Las variables que participan en el método son **a**, **i**, y **cantEliminados**. Se deben clasificar las ocurrencias de cada una de estas variables en el contexto del método **ordenarSinRep**. En la figura 4.9 se muestra el CFG del método ordenarSinRep junto con las definiciones y usos correspondientes a cada nodo.

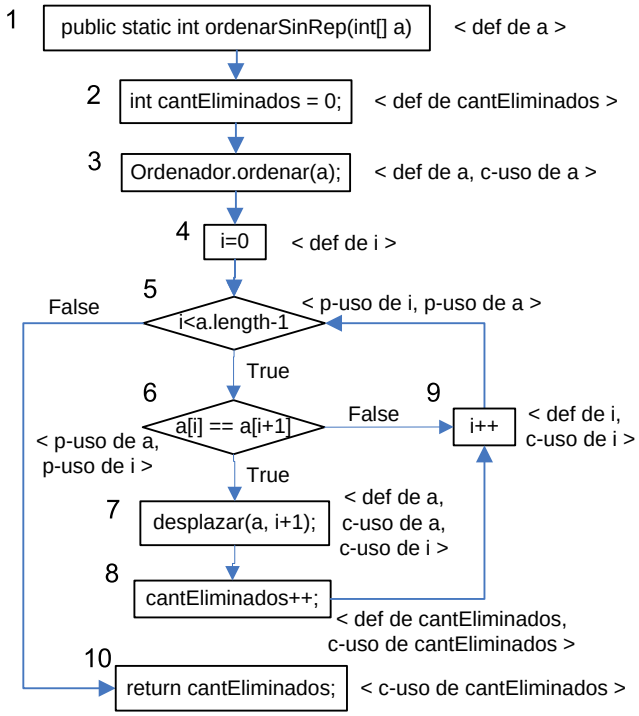



Figura 4.9: CFG de ordenarSinRep con definiciones y usos identificados

Se identifica en el nodo 1 una definición de la variable **a** de tipo array, por tratarse de un parámetro de entrada del método bajo prueba. En el nodo 2 se define la variable **cantEliminados**. En el nodo 3 se invoca al método *ordenar* pasando la variable **a** por parámetro, identificando una definición y un c-uso de **a** como se explica en 4.2.2. En el nodo 4 se identifica una definición de la variable **i**. En el nodo 5 las variables **a** e **i** son usadas en un predicado booleano que implica una bifurcación, por tanto los usos corresponden a p-usos. Lo mismo sucede en el nodo 6, donde las mismas variables **a** e **i** se usan en un predicado

booleano que implica bifurcación, correspondiendo también a p-usos. Se observa una situación similar a la del nodo 3 en el nodo 7, donde se invoca al método **desplazar** pasando las variables **a** e **i (i+1)** por parámetro. Se identifica una definición y un c-uso de **a**, y un c-uso de **i** como es explicado en 4.2.2. En el nodo 8 aparece una sentencia particular explicada en la sección 4.2.1. Esta sentencia corresponde a una definición y un c-uso de la variable **cantEliminados**. El nodo 9 es similar al 8 pero para la variable **i**, se identifica entonces una definición y un c-uso para **i**. Por último en el nodo 10 se retorna la variable **cantEliminados**. Este caso también es explicado en la sección 4.2.1 identificando un c-uso de la variable **cantEliminados**.

En este primer paso se identificaron las definiciones y usos para todas las variables en el método. Es necesario ahora identificar un camino libre-definición para cada par definición-uso, para cada variable.

Se identifican los caminos libre-definición para la variable **a**. Dicha variable es definida en los nodos 1, 3 y 7. Partiendo de la definición en el nodo 1 se llega al c-uso del nodo 3 con el camino libre-definición (1, 2, 3). Desde este nodo no existen caminos libre-definición que lleguen a los restantes usos de **a** dado que se realiza una definición en el nodo 3. Sin embargo, esto no implica un incumplimiento de la técnica, dado que no existe dicho camino. No se cumpliría con la técnica en el caso de que existiera un camino libre-definición pero no se identificara. Desde la definición del nodo 3 no existe un camino libre-definición que lleve al c-uso del nodo 3, no existe siquiera camino simple que vuelva al nodo 3. Desde el nodo 3 se llega al p-uso del nodo 5 con *True* con el camino (3, 4, 5, 6), y con *False* con el camino (3, 4, 5, 10). Desde dicha definición se alcanza al p-uso del nodo 6 con *True* mediante el camino (3, 4, 5, 6, 7), y con *False* con el camino (3, 4, 5, 6, 9). A través del camino (3, 4, 5, 6, 7) se llega al c-uso del nodo 7. Partiendo de la definición del nodo 7 no es posible alcanzar el c-uso del nodo 3. Desde el nodo 7 se llega al p-uso del nodo 5 con *True* con el camino (7, 8, 9, 5, 6), y con *False* con el camino (7, 8, 9, 5, 10). El p-uso del nodo 6 es alcanzado con *True* mediante el camino (7, 8, 9, 5, 6, 7), y con *False* con el camino (7, 8, 9, 5, 6, 9). Con el camino (7, 8, 9, 5, 6, 7) se alcanza al c-uso del nodo 7.

Se obtiene el conjunto de caminos libre-definición a ser ejecutado eliminando los caminos que se encuentran contenidos en otros. El conjunto de caminos resultante para la variable **a** es: (3, 4, 5, 10), (3, 4, 5, 6, 7), (3, 4, 5, 6, 9), (7, 8, 9, 5, 10), (7, 8, 9, 5, 6, 7) y (7, 8, 9, 5, 6, 9). Se omite mencionar el camino (1, 2, 3) ya que es ejecutado implícitamente por medio de los caminos anteriores.

Se identifican los caminos libre-definición para la variable **i**. Esta variable es definida en los nodos 4 y 9. Desde la definición del nodo 4 se llega al p-uso del nodo 5 con *True* mediante el camino (4, 5, 6), y con *False* con el camino (4, 5, 10). Desde el nodo 4 se alcanza al p-uso del nodo 6 con *True* con el camino (4, 5, 6, 7), y con *False* con (4, 5, 6, 9). Con el camino (4, 5, 6, 7) también se llega al c-uso del nodo 7, y con el camino (4, 5, 6, 9) también se alcanza el c-uso del nodo 9. Partiendo de la definición del nodo 9 se llega al p-uso del nodo 5 con *True* con el camino (9, 5, 6), y con *False* con el camino (9, 5, 10). Se identifica el camino (9, 5, 6, 7) para llegar al p-uso del nodo 6 con *True*, y el (9, 5, 6, 9) para llegar con *False*. Con el camino (9, 5, 6, 7) se alcanza al c-uso del nodo 7, y con (9, 5, 6, 9) se alcanza el c-uso del nodo 9. El conjunto de caminos resultante para la variable **i** es: (4, 5, 10), (4, 5, 6, 7), (4, 5, 6, 9), (9, 5, 10), (9, 5, 6, 7) y (9, 5, 6, 9).

Por último se identifican los caminos libre-definición para la variable **cantEliminados**. La variable es definida en los nodos 2 y 8. Desde la definición del nodo 2 se llega al c-uso del nodo 8 con el camino (2, 3, 4, 5, 6, 7, 8), y al c-uso del nodo 10 con (2, 3, 4, 5, 10). Desde la definición del nodo 8 se alcanza el c-uso del mismo nodo con el camino (8, 9, 5, 6, 7, 8), y el c-uso del nodo 10 con (8, 9, 5, 10). Entonces el conjunto de caminos libre-definición para la variable **cantEliminados** es: (2, 3, 4, 5, 6, 7, 8), (2, 3, 4, 5, 10), (8, 9, 5, 6, 7, 8) y (8, 9, 5, 10).

A partir de los caminos libre-definición identificados para todas las variables se obtiene un único conjunto de caminos que contemple a todos, eliminando los caminos contenidos en otros caminos. El conjunto de caminos libre-definición obtenido en este ejemplo es: (3, 4, 5, 6, 9), (7, 8, 9, 5, 10), (7, 8, 9, 5, 6, 7), (7, 8, 9, 5, 6, 9), (2, 3, 4, 5, 6, 7, 8), (2, 3, 4, 5, 10) y (8, 9, 5, 6, 7, 8).

Se deben identificar los valores de entrada que provoquen la ejecución de los caminos libre-definición determinados. Para lograr esto es necesario conocer el propósito de los métodos **ordenar** y **desplazar**. El método **ordenar** retorna el array de entrada ordenado de menor a mayor. El método desplazar mueve al final del array el elemento de la posición pasada por parámetro. En el cuadro 4.1 se exponen los caminos libre-definición junto con un valor de entrada para la variable **a** que provoca la ejecución del camino correspondiente. El conjunto de valores de entrada que se propone no es mínimo, por ejemplo el caso a=[3,3,3] y el caso a=[6,6,6] ejecutan lo mismo.

Cuadro 4.1: Valores de entrada para ejecutar los caminos libre-definición de Intra-método

| Camino libre-definición | Valor de entrada |
|---|---|
| (3, 4, 5, 6, 9) | a=[1,2,3] |
| (7, 8, 9, 5, 10) | a=[2,2] |
| (7, 8, 9, 5, 6, 7) | a=[3,3,3] |
| (7, 8, 9, 5, 6, 9) | a=[3,3,4] |
| (2, 3, 4, 5, 6, 7, 8) | a=[3,3,3] |
| (2, 3, 4, 5, 10) | a=[8] |
| (8, 9, 5, 6, 7, 8) | a=[6,6,6] |

Se aplica de esta forma TU a nivel de Intra-método determinando el conjunto de valores de entrada necesario para cumplir con la técnica a ese nivel.
Ahora, se aplica TU a nivel de Inter-método. En este caso para las pruebas a nivel de Inter-método se deben considerar las variables que aparecen en el método **ordenarSinRep** y que se pasan como parámetro a los métodos **ordenar** y **desplazar**. La única variable que se pasa como parámetro a esos métodos es **a**. La variable **i** no es un parámetro de entrada al método **desplazar** ya que el parámetro es el valor **i+1**. Como la variable **a** es de tipo array se deben considerar todas las definiciones y usos de la misma en los métodos **ordenar** y **desplazar**. Ambos razonamientos son expuestos en 4.2.2.

Para aplicar Inter-método se realiza el CFG' siguiendo los pasos indicados en 4.2.2 para su construcción. La figura 4.10 ilustra el grafo resultante junto con las definiciones y usos correspondientes en cada nodo.

Se identifican definiciones de **a** en los nodos 1, 11, 12 y 22, p-usos en los nodos 9, 18, 19 y 21, y c-usos en los nodos 3, 10, 11 y 22. De esta forma se

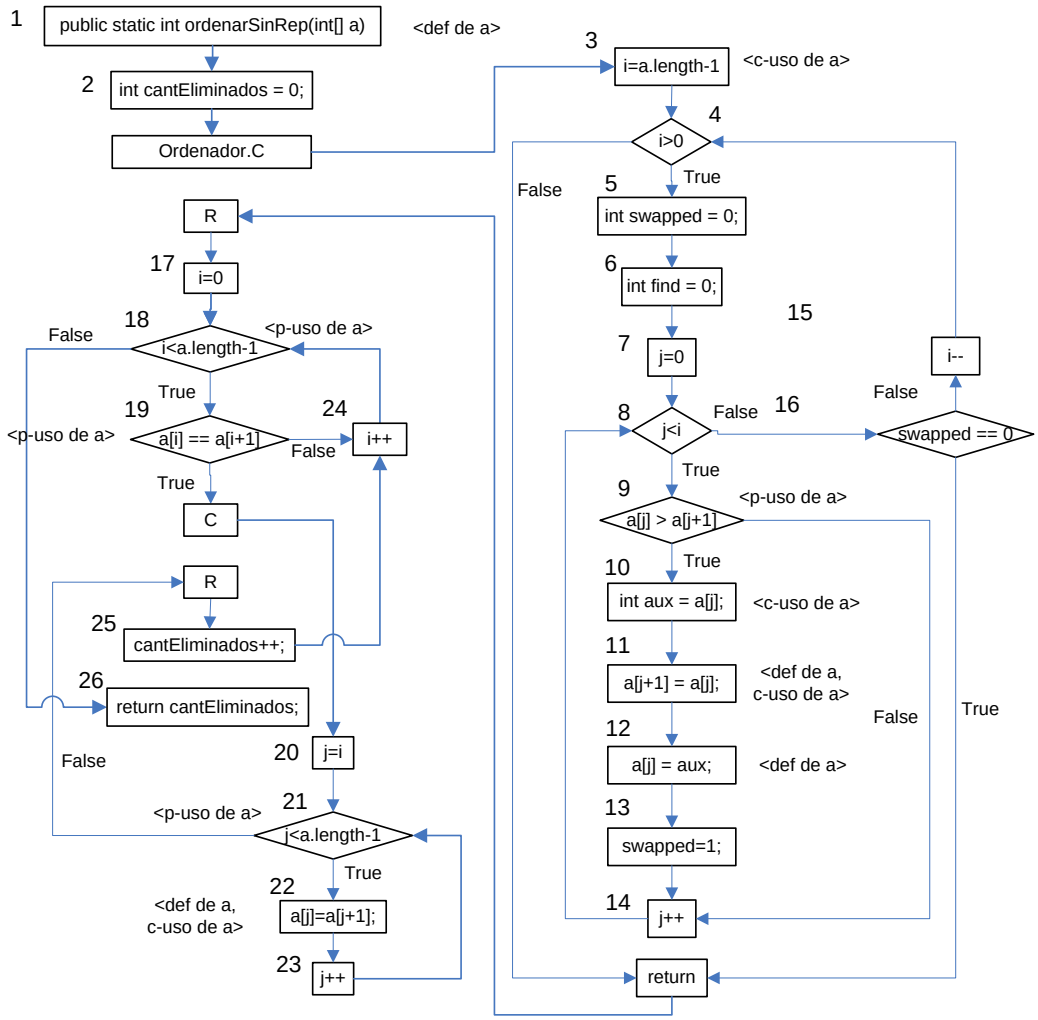



Figura 4.10: CFG' de ordenarSinRep con definiciones y usos identificados

obtienen las definiciones y usos para Inter-método.

A continuación, se identifican los caminos libre-definición para la variable **a**. Como se mencionó, dicha variable es definida en los nodos 1, 11, 12 y 22. Partiendo de la definición en el nodo 1 se llega al c-uso del nodo 3 con el camino libre-definición (1, 2, 3). Se llega a los c-usos de los nodos 10, 11 y 22 con los caminos libre-definición (1, 2, 3, 4, 5, 6, 7, 8, 9, 10), (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) y (1, 2, 3, 4, 17, 18, 19, 20, 21, 22) respectivamente. Desde el nodo 1 no existe un camino libre-definición que lleve al c-uso del nodo 12, debido a que en el nodo 11 hay una definición de **a** y no hay forma de llegar al nodo 12 sin pasar por el 11. Se llega desde el nodo 1 al p-uso del nodo 9 con *True* con el camino (1, 2, 3, 4, 5, 6, 7, 8 ,9, 10) y con *False* con el camino (1, 2, 3, 4, 5, 6, 7, 8, 9, 14). Al p-uso de 18 se llega con los caminos (1, 2, 3, 4, 17, 18, 19) y (1, 2, 3, 4, 17, 18, 26) haciendo *True* y *False* respectivamente. Se llega al p-uso de 19 desde

el nodo 1 a través de los caminos (1, 2, 3, 4, 17, 18, 19, 20) con *True* y (1, 2, 3, 4, 17, 18, 19, 24) con *False*. Por último para llegar al p-uso del nodo 21 desde el nodo 1 se logra con los caminos libre-definición (1, 2, 3, 4, 17, 18, 19, 20, 21, 22) y (1, 2, 3, 4, 17, 18, 19, 20, 21, 25) correspondientes a *True* y *False*.

Se obtuvieron los caminos libre-definición que llegan desde el nodo 1 a los c-usos y p-usos correspondientes de la variable **a**. Resta por obtener los caminos libre-definición partiendo de las definiciones de **a** de los nodos 11, 12 y 22. Para simplificar el análisis se listan dichos caminos libre-definición. Desde la definición de **a** del nodo 11 no existe ningún camino libre-definición que lleve a sus usos debido a que en el siguiente nodo (nodo 12) hay una nueva definición de **a**.

Partiendo de la definición del nodo 12 se llega a los usos con los caminos libre-definición (12, 13, 14, 8, 9, 10), (12, 13, 14, 8, 9, 10, 11) y (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 22), y a los p-usos con los caminos (12, 13, 14, 8, 9, 10), (12, 13, 14, 8, 9, 14), (12, 13, 14, 8, 16, 17, 18, 19), (12, 13, 14, 8, 16, 17, 18, 26), (12, 13, 14, 8, 16, 17, 18, 19, 20), (12, 13, 14, 8, 16, 17, 18, 19, 24), (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 22) y (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 25). Al c-uso del nodo 3 no es posible llegar dado que dicho nodo es inalcanzable desde el nodo 12.

Desde la definición del nodo 22 son alcanzables los p-usos del nodo 18, 19 y 21 y el c-uso del nodo 22. Los caminos libre-definición con los que se llega a dichos p-usos y al c-uso son (22, 23, 21, 25, 24, 18, 19), (22, 23, 21, 25, 24, 18, 26), (22, 23, 21, 25, 24, 18, 19, 20), (22, 23, 21, 25, 24, 18, 19, 24), (22, 23, 21, 25) y (22, 23, 21, 22).

A partir de los caminos libre-definición identificados para la variable **a** se obtiene un único conjunto de caminos que contempla a todos, eliminando los caminos contenidos en otros. El conjunto de caminos libre-definición obtenido para inter-método es: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11), (1, 2, 3, 4, 17, 18, 19, 20, 21, 22), (1, 2, 3, 4, 5, 6, 7, 8, 9, 14), (1, 2, 3, 4, 17, 18, 26), (1, 2, 3, 4, 17, 18, 19, 24), (1, 2, 3, 4, 17, 18, 19, 20, 21, 25), (12, 13, 14, 8, 9, 10, 11), (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 22), (12, 13, 14, 8, 9, 14), (12, 13, 14, 8, 16, 17, 18, 26), (12, 13, 14, 8, 16, 17, 18, 19, 24), (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 22), (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 25), (22, 23, 21, 25, 24, 18, 26), (22, 23, 21, 25, 24, 18, 19, 20), (22, 23, 21, 25, 24, 18, 19, 24) y (22, 23, 21, 22).

Se identifica el conjunto de valores de entrada que provoca la ejecución de los caminos libre-definición encontrados para Inter-método. En el cuadro 4.2 se detallan los caminos libre-definición necesarios para cumplir con la técnica junto con un valor de entrada que provoca la ejecución de cada camino. En algunos casos se observa que no es posible ejecutar el camino libre-definición para ningún valor de entrada debido a la secuencia de instrucciones que ocurren en el programa.

Es importante notar que algunos de los valores de entrada determinados para Intra-método ocasionen también la ejecución de caminos identificados para Inter-método, lo cual reduce la cantidad de casos de prueba diseñados. Unificando los valores de entrada que ejecutan los caminos libre-definición para Intra-método e Inter-método, y estableciendo los resultados esperados se completa la construcción de los casos de prueba para TU. El cuadro 4.3 presenta el conjunto de casos de prueba resultante. Como se explicó en el capítulo 6, el método OrdenarSinRep ordena un array **a** de menor a mayor y sin repetidos desde la posición 0 hasta la posición a.length - la cantidad de elementos repetidos - 1. Como resultado de este ordenamiento, desde la posición a.length - la cantidad

Cuadro 4.2: Valores de entrada para ejecutar los caminos libre-definición de Inter-método

| Camino libre-definición | Valor de entrada |
|---|---|
| (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) | a=[3,2,1] |
| (1, 2, 3, 4, 17, 18, 19, 20, 21, 22) | a=[8] |
| (1, 2, 3, 4, 5, 6, 7, 8, 9, 14) | a=[1,2,3] |
| (1, 2, 3, 4, 17, 18, 26) | a=[8] |
| (1, 2, 3, 4, 17, 18, 19, 24) | Camino no ejecutable |
| (1, 2, 3, 4, 17, 18, 19, 20, 21, 25) | Camino no ejecutable |
| (12, 13, 14, 8, 9, 10, 11) | a=[3,2,1] |
| (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 22) | Camino no ejecutable |
| (12, 13, 14, 8, 9, 14) | a=[3,2,8] |
| (12, 13, 14, 8, 16, 17, 18, 26) | Camino no ejecutable |
| (12, 13, 14, 8, 16, 17, 18, 19, 24) | Camino no ejecutable |
| (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 22) | Camino no ejecutable |
| (12, 13, 14, 8, 16, 17, 18, 19, 20, 21, 25) | Camino no ejecutable |
| (22, 23, 21, 25, 24, 18, 26) | a=[2,2] |
| (22, 23, 21, 25, 24, 18, 19, 20) | Camino no ejecutable |
| (22, 23, 21, 25, 24, 18, 19, 24) | Camino no ejecutable |
| (22, 23, 21, 22) | a=[3,3,4] |

de elementos repetidos hasta a.length - 1 se desconocen los valores de **a**. Estos valores desconocidos son representados con un ? en el cuadro 4.3.

Cuadro 4.3: Conjunto de casos de prueba para cumplir con TU

| Valor de entrada | Resultado esperado |
|---|---|
| a=[1,2,3] | a=[1,2,3] |
| a=[2,2] | a=[2,?] |
| a=[3,3,3] | a=[3,?,?] |
| a=[3,3,4] | a=[3,4,?] |
| a=[3,3,3] | a=[3,?,?] |
| a=[8] | a=[8] |
| a=[6,6,6] | a=[6,?,?] |
| a=[3,2,1] | a=[1,2,3] |
| a=[3,2,8] | a=[2,3,8] |

En resumen, se ha mostrado un ejemplo de aplicación de TU a niveles de Intra-método e Inter-método. Este ejemplo fue construido nodo por nodo y camino a camino, sin embargo, la aplicación de la técnica puede realizarse de forma más rápida. A medida que el verificador va obteniendo experiencia puede aprovechar y obtener desde una definición un camino libre-definición que llegue a varios usos, disminuyendo de esta forma la cantidad de casos de prueba.

Los pasos que pueden guiar al verificador para aplicar la técnica son:

1. Identificar las definiciones y usos de las variables involucradas.
2. Hallar los caminos libre-definición para cada variable.
3. Generar los casos de prueba para cumplir con la técnica.

### 4.2.4. Anomalías en el Flujo de Datos

Para finalizar la sección de DFT se mencionan las anomalías que pueden producirse en el flujo de datos. Las anomalías son defectos en el código producidos por un flujo de datos incorrecto. Es posible detectarlos durante la etapa de diseño de los casos de prueba. Pueden deberse, por ejemplo, a equivocaciones en los nombres de las variables, omisión de sentencias, y uso incorrecto de parámetros.

Ejemplos:

- dd : definiciones consecutivas: es el caso en el que se realizan dos definiciones de una misma variable sin hacer ningún uso de la misma entre dichas definiciones. Excluimos de esta anomalía dos definiciones consecutivas de un array.

- d- : definición sin posterior uso: es el caso en el que se realiza una definición de una variable sin hacer ningún uso posterior de la misma.

- -u : uso sin previa definición: es el caso en el que se realiza un uso de una variable sin hacer ninguna definición previa de la misma.

# Capítulo 5

# Introducción al Experimento Formal

Si bien las pruebas unitarias ya están fuertemente establecidas en la industria de software aún no se conoce qué tan efectiva es cada técnica de verificación. Tampoco se conoce el costo asociado o si la efectividad varía según los distintos tipos de defecto. En definitiva, al momento de realizar pruebas unitarias la decisión acerca de cuál o cuáles técnicas usar no es trivial. Obtener este conocimiento sobre las técnicas no es sencillo. No es posible demostrar formalmente cuál es la efectividad y el costo de una técnica de verificación. Además, la forma en la cual se comporta la técnica varía según la persona que la esté ejecutando, el programa sobre el cual se está ejecutando, el lenguaje de programación, entre otros.

Debido a que aún no se conoce con certeza el comportamiento de las técnicas de pruebas, en la Facultad de Ingeniería de la Universidad de la República se han realizado varios experimentos formales para obtener datos más precisos en este sentido. Actualmente se han culminado 3 experimentos. El primer experimento consta de un programa pequeño y simple sobre el que se estudiaron 4 técnicas de verificación: inspección de escritorio, partición en clases de equivalencia y valores límite, tablas de decisión y criterio de cubrimiento de condición múltiple. El segundo experimento, que denominamos y citamos de aquí en más como Experimento Año 2008, examinó las mismas técnicas de verificación sobre 4 programas, dichos programas son más complejos que los encontrados en la literatura de experimentos formales para técnicas de verificación. Estos experimentos se presentan en [21] y [19] respectivamente. El tercer experimento es el realizado en este proyecto.

## 5.1. Estructura del Experimento

El proyecto tiene varios objetivos, entre ellos obtener una idea de la efectividad y el costo de dos técnicas de verificación: Cubrimiento de Sentencias (CS) y Todos los Usos (TU). Para alcanzar este objetivo se llevó a cabo un experimento formal. Como primer paso fue necesario estudiar a fondo dichas técnicas y definir los criterios necesarios para especificar correctamente su aplicación. Las técnicas se encuentran detalladas en el capítulo 4.

Para poder realizar el experimento fue necesario contar con un conjunto de artefactos que brindaran soporte al mismo. Entre ellos se encuentran las unidades experimentales, en este caso los programas sobre los cuales aplicar las técnicas estudiadas. Fue necesario especificar el proceso de verificación a ser seguido por los sujetos, para ello se utilizó una **Guía de Verificación**. Debió contarse con un medio para registrar los datos obtenidos durante la verificación, para ello se utilizó una planilla excel que denominamos **Grillo**. Además, los sujetos fueron capacitados en la aplicación de las técnicas, en el uso de la guía y la planilla de registro, para ello se prepararon y dictaron clases niveladoras. Todos estos materiales de apoyo son presentados en la sección 5.2.

En la figura 5.1 se muestran las fases que componen a cualquier experimento formal. Nuestro experimento respeta dicha composición en fases.

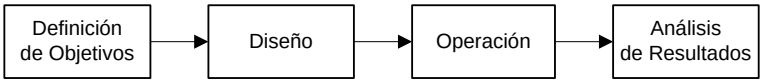



Figura 5.1: Fases de un Experimento Formal

La primer fase es la de definición, en donde se define el experimento en términos de sus objetivos, y se establecen las variables a ser examinadas. La siguiente fase corresponde al diseño, en la cual se determina el diseño del experimento, es decir que se planifica la forma en la que va a ser ejecutado el experimento, indicando las condiciones exactas bajo las cuales va a ser conducido. En la fase de operación se ejecuta el diseño del experimento, esto implica la ejecución de cada experimento unitario correspondiente al diseño elegido. En nuestro caso la ejecución del experimento consistió en la aplicación de las técnicas CS y TU por parte de un conjunto de sujetos a un programa escrito en Java. Para poder realizar esta ejecución fue necesario llevar a cabo el entrenamiento de los sujetos respecto a la aplicación de las técnicas y al uso de los materiales de soporte. El entrenamiento fue dividido en dos instancias que denominamos experiencias I y II. Por último, en la fase de análisis de resultados se analizaron los datos recolectados durante la ejecución. El experimento es detallado en el capítulo 7.

La figura 5.2 corresponde a las fases de nuestro experimento, en donde se ilustra la división de la fase de Operación en tres sub-fases.

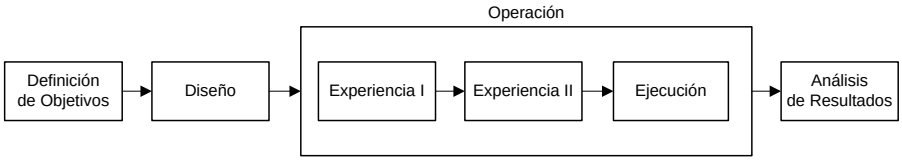



Figura 5.2: Fases de Nuestro Experimento

Las sub-fases denominadas experiencias I y II (llamadas también experiencias iniciales) corresponden a la capacitación y entrenamiento que se brindó a los sujetos. Son dos experiencias ya que se entrenó a los sujetos divididos en dos grupos, con cierta separación temporal entre el entrenamiento del primer grupo y el entrenamiento del segundo. La experiencia I también fue muy útil para adquirir información valiosa sobre la adecuación de la capacitación brindada a

los estudiantes, y mejorarla para la experiencia II. Además, otorgó una visión general de las dificultades afrontadas en la aplicación de las técnicas.

En estas primeras experiencias se busca obtener una idea del costo de las técnicas utilizadas para poder estimar el tiempo que requiere realizar la ejecución del experimento. Son experiencias de pequeño porte que consistieron en la aplicación de las técnicas de verificación CS y TU por parte de los sujetos a un simple programa escrito en Java. La duración de cada experiencia fue de aproximadamente 3 semanas. En la experiencia I participaron 10 sujetos, de los cuales 5 aplicaron TU y 5 aplicaron CS. En la experiencia II participaron 11 sujetos, donde 6 aplicaron TU y 5 aplicaron CS. Todos ellos son estudiantes de la carrera Ingeniería en Computación de la Facultad de Ingeniería. Son estudiantes avanzados dado que todos se encuentran en cuarto o quinto año.

El diseño del experimento corresponde a un diseño de un factor (técnica de verificación) con dos alternativas (CS y TU). Los sujetos que participaron en la ejecución del experimento son un subconjunto de los sujetos que participan en las experiencias iniciales; 4 estudiantes de la experiencia I y 10 de la experiencia II. De estos 14 sujetos, 8 aplicaron TU y 6 aplicaron CS durante un período de 8 semanas.

El experimento fue de mayor porte que las experiencias iniciales con respecto al programa verificado. El programa de las experiencias iniciales consta de dos clases Java, de 18 LOCs y 19 LOCs respectivamente. Su función es la de ordenar una serie de enteros de un array y eliminar elementos duplicados. El programa utilizado en el experimento tiene un tamaño aproximado de 1820 LOCs distribuidos en 14 clases Java. El propósito general del programa es la liquidación de sueldos para funcionarios docentes y no docentes de una facultad ficticia. El programa del experimento es más realista que el programa utilizado en el entrenamiento respecto a la práctica de la industria.

Si bien las experiencias iniciales proveen de la capacitación necesaria para realizar la ejecución del experimento, se considera que las mismas constituyen un experimento en sí mismo. Cada experiencia inicial tiene su propio diseño y su propia operación. Dado que ambos experimentos (experiencias iniciales) poseen características muy similares es posible realizar un análisis de resultados en conjunto. Ambas experiencias iniciales se presentan como un experimento en el capítulo 6.

La figura 5.3 ilustra la formación de las experiencias y experimentos explicados.

## 5.2. Materiales de Soporte al Experimento

En esta sección se presentan los artefactos que brindaron soporte a la realización del experimento.

### 5.2.1. Programas

Los programas utilizados en las experiencias iniciales y en el experimento grande son diferentes. El programa utilizado en las experiencias iniciales es sencillo debido a que su función es la de ordenar una serie de enteros y eliminar elementos duplicados. Sólo consta de dos clases, una de ellas cuenta con 18 LOCs y la otra con 19 LOCs, ambas sin comentarios. Este programa es detallado en

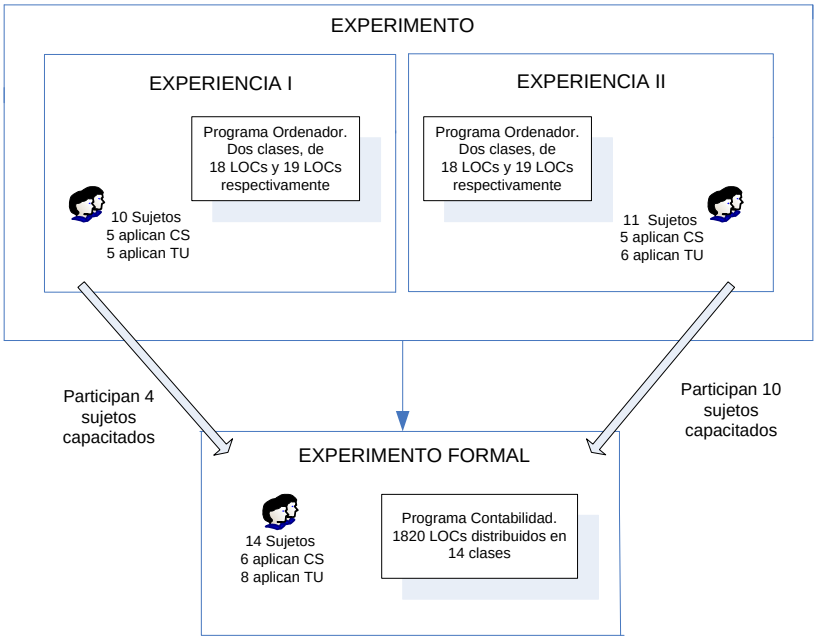



Figura 5.3: Diagrama explicativo de experimentos

la sección 6.1. El programa utilizado en el experimento grande, denominado Contabilidad, es de mayor tamaño que el utilizado en las experiencias iniciales. Además, es de mayor complejidad que los programas encontrados en la literatura de experimentos formales para técnicas de verificación. El propósito general del programa es la liquidación de sueldos para funcionarios docentes y no docentes de una facultad ficticia. Tiene un tamaño aproximado de 1820 LOCs (sin comentarios) distribuidos en 14 clases. Este programa se detalla en la sección 7.1.

### 5.2.2. Guía

La guía fue construida en base a la utilizada en el primer experimento realizado en la Facultad de Ingeniería. Se utiliza de la guía original únicamente la sección correspondiente a las técnicas de caja blanca. Además, se realizan los ajustes y agregados necesarios para este experimento.

En la guía se presentan las fases que intervienen en el proceso de verificación dinámica con caja blanca. El proceso comprende 3 fases: Preparación, Diseño y Ejecución.

Durante la fase de Preparación se realiza un chequeo inicial garantizando que se puede llevar a cabo la verificación requerida. Se comprueba que se dispone de los archivos fuentes a probar, un archivo con el diagrama de colaboración, el javadoc de las clases, y la planilla Grillo de registro de defectos y tiempos.

En la fase de Diseño se realiza el diseño de los casos de prueba cumpliendo con la técnica de verificación a aplicar. Se deben diseñar los casos de forma *bottom-up*. Para cada método se define el conjunto de valores de entrada que

cumplen con la técnica, y la salida esperada (o comportamiento esperado) para dichos valores de entrada, de manera de conformar los casos de prueba. Luego se debe eliminar los casos de prueba que no puedan ser ejecutados (caminos imposibles, etc.), y por último codificar los casos de prueba diseñados en JUnit. En esta fase se registra el tiempo insumido en el diseño y codificación de los casos de prueba en la planilla Grillo.

Los defectos que sean encontrados durante la fase de diseño deben ser registrados como se explica en la fase de Ejecución. En este caso el tiempo de detección del defecto es 0 (cero).

Durante la fase de Ejecución se realiza la ejecución de los casos de prueba diseñados. Los casos se deben ejecutar de forma *bottom-up*. La fase finaliza cuando no existen casos de prueba que fallan. Mientras existan casos de prueba que fallan se debe:

- Escoger un caso de prueba que falla.
- Inicializar el tiempo de búsqueda del defecto.
- Buscar en el programa el defecto que causa la falla. Se puede usar una herramienta de *debug* y ejecutar el caso de prueba.
- Detener el tiempo de búsqueda cuando se encuentra el defecto.
- Para cada defecto encontrado registrar los datos solicitados en la planilla Grillo.
- Solicitar la corrección del defecto.
- Correr nuevamente el caso de prueba para ver si ya no falla.

La guía completa puede consultarse en el Anexo B.

### 5.2.3. Grillo

El registro de los defectos por parte de los sujetos durante el experimento permite tener un control y seguimiento de los mismos. Los defectos encontrados se registran en una planilla excel denominada Grillo. Esta planilla fue construida con el propósito de mantener el registro de todos los defectos encontrados por los sujetos junto con su tiempo de detección, tiempo de diseño de los casos de prueba, y tiempo de ejecución.

La planilla fue adaptada a cada experimento (experiencias iniciales y experimento grande) teniendo en cuenta las clases a probar en cada uno. La planilla contiene dos lenguetas, un ejemplo de la primera se ilustra en la figura 5.4. En esta lengueta se ingresan el nombre de la experiencia, el nombre del sujeto que verifica, el nombre del producto o programa a verificar, la técnica que se aplica, las fechas de comienzo y fin, y los tiempos de diseño y ejecución. El tiempo de diseño se ingresa por cada clase que compone el programa y corresponde al tiempo que le lleva al sujeto diseñar y codificar los casos de prueba. El tiempo de ejecución corresponde al tiempo que insume ejecutar los casos de prueba. Este tiempo se registra solo para la clase main del programa Contabilidad, ya que es la encargada de implementar la interfaz con el usuario, y por lo tanto, requiere


Figura 5.4: Grillo - Primer lengueta

interacción con el mismo. Para los casos de prueba codificados en JUnit no se registra el tiempo de ejecución ya que se considera nulo.

Un ejemplo de la segunda lengueta se ilustra en la figura 5.5. En esta lengueta se ingresa una fila para cada defecto encontrado. En cada fila se ingresa el número de la línea de código donde se encuentra el defecto y una descripción del mismo. También se ingresa el nombre de la clase que contiene el defecto y el tiempo que transcurre entre el inicio de la búsqueda del defecto y el momento de detección del mismo. Se indica además la menor estructura de código dentro de la cual se encuentra el defecto (p.e. WHILE, IF, METODO) y la línea de comienzo de la estructura. Por último se especifica si el defecto fue encontrado durante el diseño o durante la ejecución de los casos de prueba.


Figura 5.5: Grillo - Segunda lengueta

### 5.2.4. Clases de Capacitación

Para capacitar a los sujetos en la aplicación de las técnicas y en el uso de la Guía y el Grillo se prepararon y dictaron clases teórico-prácticas. Las mismas fueron preparadas y dictadas por las autoras de este proyecto, contando con la supervisión del tutor. Se dictó la misma clase en dos instancias, para capacitar a los sujetos divididos en dos grupos. Cada clase tuvo un día de duración.

En la primer mitad del día se proporcionó la clase teórica, en la cual se realiza un repaso de algunos conceptos referentes a Ingeniería de Software y se explican las técnicas a utilizar. Para exponer los conceptos teóricos se crearon diapositivas, las mismas pueden consultarse en el Anexo A. También se presentan la Guía y el Grillo.

Durante la segunda mitad del día se realizan ejercicios prácticos en forma individual y grupal con el objetivo de fijar los conceptos teóricos. En dichos ejercicios se aplican las técnicas CS y TU sobre fragmentos de programas desarrollados en Java.

Como actividad de integración se realiza un almuerzo compartido al mediodía.

# Capítulo 6

# Experiencias Iniciales

En este capítulo presentamos las experiencias I y II. Estas son consideradas como un experimento en sí mismo, dividido en dos instancias.

Ambas experiencias se estructuran de igual forma, y consisten en la aplicación de dos técnicas de verificación por parte de dos grupos de estudiantes a un simple programa escrito en Java. Las técnicas de verificación son Cubrimiento de Sentencias (CS) y Todos los Usos (TU) que se encuentran detalladas en el capítulo 4.

Las variables de respuesta consideradas en este experimento son el costo y la efectividad de las técnicas. El costo es entendido como el tiempo que requiere la aplicación de la técnica. La efectividad se define como el porcentaje de defectos encontrados.

La hipótesis nula de efectividad, hipótesis que se quiere rechazar, plantea que las medianas de la efectividad de las técnicas son iguales. La hipótesis nula de costo plantea que las medianas de costo de las técnicas son iguales. Las hipótesis alternativas correspondientes simplemente indican que las medianas son diferentes.

El resto del capítulo se estructura de la siguiente forma. En la sección 6.1 se presenta el programa utilizado en ambas experiencias (especificación y código fuente), y en la sección 6.2 se muestran los defectos contenidos en el código. Las características de los sujetos se describen en la sección 6.3. El diseño elegido se detalla en 6.4 y la operación del experimento en 6.5. Los resultados y las conclusiones obtenidas se describen en 6.7 y 6.8 respectivamente.

## 6.1. Programa

El programa utilizado en las experiencias iniciales es sencillo, simple y escrito en Java. Consideramos que es sencillo debido a que su función es la de ordenar una serie de enteros y eliminar la duplicación de cada elemento. Además, sólo consta de dos clases, una de ellas cuenta con 18 LOCs y la otra con 19 LOCs, ambas sin comentarios. La interacción entre las clases también es sencilla: la clase OrdenadorSinRep invoca a un método de la clase Ordenador para que el array sea ordenado antes de que se eliminen los elementos repetidos.

#### Especificaciones y Código Fuente

La Figura 6.1 muestra el diagrama de colaboración de las dos clases del programa. Cada clase tiene sólo un método público, acompañado de su especificación.

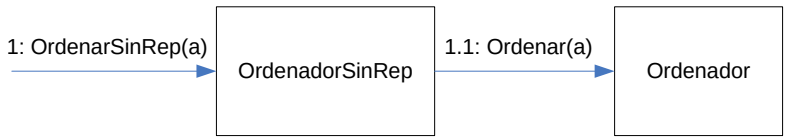



Figura 6.1: UML Diagrama de Colaboración del Programa

A continuación se presenta la firma y la especificación del método Ordenar de la clase Ordenador.

---

**public static void ordenar(int[] a)**

El método retorna el array ordenado de menor a mayor. En caso que el array sea nulo o vacío permanece incambiado.

Por ejemplo: El array de entrada es: **a** = [1, 3, 5, 3, 3]. Después de ejecutar el método el array retornado es: [1, 3, 3, 3, 5].

a: parámetro de entrada que contiene los enteros a ordenar.

```
1 public class Ordenador {
2
3    public static void ordenar (int[] a){
4       for(int i=a.length-1; i>0; i--){
5          int intercambia = 0;
6          int encontre = 0;
7          for (int j=0; j<i; j++){
8             if (a[j] > a[j+1]){
9                int aux = a[j];
10               a[j+1] = a[j];
11               a[j] = aux;
12               intercambia=1;
13            }
14         }
15         if (intercambia == 0) {
16            return;
17         }
18      }
19   }
20}
```

---

A continuación se presenta la firma y la especificación del método OrdenarSinRep de la clase OrdenadorSinRep.

---

**public static int OrdenarSinRep(int[] a)**

El método retorna el array **a** ordenado de menor a mayor y sin repetidos desde la posición 0 hasta la posición a.length - la cantidad de elementos repetidos - 1.

En el array **a**, desde la posición a.length - la cantidad de elementos repetidos hasta a.length - 1 se desconocen los valores de **a** (es decir, no importan). Ejemplo: **a** = [5, 4, 5, 6, 6, 5] Cantidad de elementos repetidos = 3. El número 5 se repite dos veces y el 6 una. El array **a** luego de ejecutarse el método desde la posición cero hasta la posición 2 es [4, 5, 6]. La posición 2 es calculada como 6-3-1. Desde la posición 3 hasta la 5 (a.length-1) se desconocen los valores de **a**. En caso que el array sea null o vacío se devuelve cero en la cantidad de eliminados y **a** sigue siendo null o vacío según el caso.

```
1 public class OrdenadorSinRep {
2
3    public static int ordenarSinRep(int[] a){
4       int cantElim = 0;
5       Ordenador.ordenar(a);
6       for(int i=0; i<a.length-1; i++){
7          if (a[i] == a[i+1]) {
8             desplazar(a, i+1);
9             cantElim++;
10         }
11      }
12      return cantElim;
13   }
14
15   private static void desplazar(int[] a, int i){
16      for(int j=i; j<a.length-1; j++){
17         a[j]=a[j+1];
18      }
19   }
20}
```

---

## 6.2. Defectos del Programa

En esta sección se presentan los defectos contenidos en el código que son relevantes para el análisis.

Los defectos se clasifican como Posible Falla (PF) o No Falla (NF). Los defectos de PF son los que pueden producir una falla durante la ejecución del programa. Los defectos NF nunca producen una falla durante la ejecución, sin embargo, pueden causar otros problemas, por ejemplo, problemas de rendimiento o problemas durante la fase de mantenimiento del software.

La clase Ordenador tiene 7 defectos a considerar. Ellos se nombran con las letras mayúsculas de la **A** a la letra **G**. En la clase OrdenadorSinRep 6 defectos son analizados, que son nombrados con letras minúsculas de la **a** a la **f**.

### 6.2.1. Defectos de la clase Ordenador

Aquí se presentan los defectos de la clase Ordenador.

La figura 6.2 muestra los defectos en el código con una elipse alrededor de ellos. A continuación se presenta la descripción de cada defecto.

```
1 public class Ordenador {
2   X                                          //Falta constructor "private Ordenador(){}" - Defecto C
3   public static void ordenar (int[] a){  //Nombre no nemotécnico para la variable a – Defecto E
4     for(int i=a.length-1; i>0; i--){     //Acceso incorrecto si a es nulo – Defecto A
5       int intercambia = 0;                  //Variable intercambia debe ser booleana – Defecto D
6       int encontre = 0;                        //encontre no se usa nunca – Defecto G
7       for (int j=0; j<i; j++){
8         if (a[j] > a[j+1]){
9           int aux = a[j];              //Swap incorrecto. Debe ser j +1 en lugar de j – Defecto B
10          a[j+1] = a[j];
11          a[j] = aux;
12          intercambia=1;
13        }
14      }
15      if (intercambia == 0) {
16        return;                        //Corta el loop – Defecto F
17      }
18    }
19  }
20 }
```

Figura 6.2: Defectos de la clase Ordenador

**Defecto A - PF**

El método Ordenar comienza con una sentencia for en la línea 4. Esta sentencia accede al array a través de la operación que retorna su largo. Si el arreglo es nulo durante la ejecución se produce una falla y el programa termina abruptamente. El defecto consiste en no chequear que el array sea distinto de *null* antes de acceder al mismo.

**Defecto B - PF**

El método Ordenar realiza un *swap* entre variables del array, esto ocurre desde la línea 9 a la 11 en el código. El swap es incorrecto porque el valor que contiene el array en la posición [j+1] no se conserva. Una falla debida a este defecto se muestra en la Figura 6.3. La figura presenta el estado del array **a** luego de la ejecución del for anidado.

```
a = [1, 3, 5, 3, 3]      Resultado Esperado= [1, 3, 3, 3, 5]

i = 4, j = 0, a[0] <= a[1]  =>  a = [1, 3, 5, 3, 3]
i = 4, j = 1, a[1] <= a[2]  =>  a = [1, 3, 5, 3, 3]
i = 4, j = 2, a[2]  >  a[3]  =>  a = [1, 3, 5, 5, 3] //Falla debida a B
i = 4, j = 3, a[3]  >  a[4]  =>  a = [1, 3, 5, 5, 5] //Falla debida a B
i = 3
El array está en orden, por lo que no se realizan más swaps.
Línea 16 provoca que el método retorne.

Resultado Obtenido = [1, 3, 5, 5, 5]
```

Figura 6.3: Una ejecución del método Ordenar que muestra la ocurrencia del defecto B

**Defecto C - NF**

La clase tiene un único método y el mismo es estático. No es razonable construir un objeto de esta clase. Si el compilador de Java no encuentra un

constructor crea automáticamente un constructor público por defecto (sin parámetros), permitiendo la creación de objetos de esta clase. Un constructor privado es necesario para evitar esto.

**Defecto D - NF**

La variable del *swap* debe ser booleana pero se define como un entero. La variable se define en la línea 6.

**Defecto E - NF**

El nombre de la variable para el array **a** no es nemotécnico. Reemplazar el nombre afecta varias líneas de código.

**Defecto F - NF**

El método tiene dos bucles, el exterior comienza en la línea 4. La línea 15 chequea el valor de intercambia y en caso de que es cero (no se han realizado cambios en el bucle interno) el método retorna. Este defecto puede removerse agregando la condición al bucle externo. Este defecto es particular porque no podía ser considerado un defecto en sí mismo.

**Defecto G - NF**

En la línea 6 se define la variable encontre y no se usa nunca en el programa.

### 6.2.2. Defectos de la clase OrdenadorSinRep

Aquí se presentan los defectos de la clase OrdenadorSinRep

La Figura 6.4 muestra los defectos en el código con una elipse alrededor de ellos. A continuación se describe cada defecto.

```
1  public class OrdenadorSinRep {
2  X                                          //Falta Constructor "private OrdenadorSinRep(){}" - Defecto c
3      public static int ordenarSinRep(int[] a){  //Nombre no nemotécnico para la variable a – Defecto e
4          int cantElim = 0;
5          Ordenador.ordenar(a);
6          for(int i=0; i<a.length-1; i++){    //Acceso errónea a la variable a y condición del for incorrecta – Defecto d
7              if (a[i] == a[i+1]) {
8                  desplazar(a, i+1);          //Variable i se incrementa incorrectamente despúes de este bloque – Defecto b
9                  cantElim++;
10             }
11         }
12         return cantElim;
13     }
14
15     private static void desplazar(int[] a, int i){
16         for(int j=i; j<a.length-1; j++){          //Condición errónea en for – Defecto f
17             a[j]=a[j+1];
18         }
19     }
20 }
```

Figura 6.4: Defectos de la clase OrdenadorSinRep

**Defecto a - PF**

El defecto es similar al defecto **A**.

**Defecto b - PF**

Después de llamar al método desplazar el índice **i** se incrementa en uno, si hay más de 2 enteros iguales en el array esto produce una falla porque algunos de ellos no son considerados. Una ejecución que muestra este defecto, el defecto **d**, y sus fallas asociadas se muestra en la Figura 6.5. El signo de interrogación en el resultado esperado significa que el valor en esta posición no importa. Este defecto puede ser removido de diferentes maneras. Una forma sencilla pero no

muy buena es el decremento de **i** después de llamar al método desplazar. Una mejor solución es añadir un bucle hasta encontrar un entero diferente.

```
a = [1, 3, 3, 3, 5]
Resultado Esperado = [1, 3, 5,  ?, ?]    Retorno 2

i = 0, a[0] !=  a[1]  =>  a = [1, 3, 3, 3, 5], cantElim = 0
i = 1, a[1] == a[2]  =>  a = [1, 3, 3, 5, 5], cantElim = 1  //Ejecución del defecto b
i = 2, a[2] !=  a[3]  =>  a = [1, 3, 3, 5, 5], cantElim = 1  //Falla debido a b
i = 3, a[3] == a[4]  =>  a = [1, 3, 3, 5, 5], cantElim = 2  //Falla debido a d
i = 4 => el método termina

La falla causada por el defecto d provoca que el método retorne el valor correcto
de cantElim

Resultado Obtenido= [1, 3, 3, 5, 5]       Retorno 2
```

Figura 6.5: Una ejecución del método OrdenarSinRep que muestra la ocurrencia de los defectos b y d

**Defecto c - NF**

El defecto es similar al defecto **C**.

**Defecto d - PF**

Cuando se encuentran elementos iguales el método desplazar es ejecutado, y este traslada a los elementos repetidos hacia el final del array como efecto secundario. Estos últimos elementos no son de importancia y la especificación es clara al respecto. Sin embargo, estos elementos repetidos son considerados como elementos iguales causando un resultado incorrecto en el método OrdenarSinRep. El defecto puede ser removido cambiando la línea 6 del método:

for(int i=0; i<a.length-1; i++)

por la línea

for(int i=0; i<a.length-1-cantElim; i++)

La figura 6.5 muestra este defecto junto al defecto **b** causando fallas durante la ejecución. La falla debida al defecto **d** afecta al contador de elementos repetidos. Otro ejemplo mostrando una falla en los resultados (el array y el contador) se muestra en la Figura 6.6.

```
a = [1, 3, 3, 3, 3, 5]
Resultado Esperado = [1, 3, 5,  ?, ?, ?]    Retorno 2

i = 0, a[0] !=  a[1]  =>  a = [1, 3, 3, 3, 3, 5],  cantElim = 0
i = 1, a[1] == a[2]  =>  a = [1, 3, 3, 3, 5, 5],  cantElim = 1  //Falla debida a b
i = 2, a[2] == a[3]  =>  a = [1, 3, 3, 5, 5, 5],  cantElim = 2  //Falla debida a b
i = 3, a[3] == a[4]  =>  a = [1, 3, 3, 5, 5, 5],  cantElim = 3  //Falla debida a d
i = 4, a[4] == a[5]  =>  a = [1, 3, 3, 5, 5, 5],  cantElim = 4  //Falla debida a d
i = 5 => el método termina

Resultado Obtenido = [1, 3, 3, 5, 5, 5]       Retorno 4
```

Figura 6.6: Otra ejecución que presenta las ocurrencias de los defectos b y d

**Defecto e - NF**

El defecto es similar al defecto **E**.

**Defecto f - NF**

El método desplazar tiene un for en la línea 16 que recorre el array de una posición inicial (que es pasada al método) hasta la posición final. No es necesario recorrer el array hasta la última posición ya que al final hay elementos que no deben ser considerados. Este es un defecto que afecta al rendimiento pero no los resultados. Para optimizar el método la línea mencionada se puede sustituir por la siguiente:

for (int j = i, j <a.length-1-cantElim; j++) (

La variable cantElim debe ser pasada como parámetro al método.

En el Cuadro 6.1 se presenta la cantidad de defectos discriminados por clase, tipo de defecto (PF, NF), y los totales.

Cuadro 6.1: Cantidad de defectos por clase, tipo, y totales

| Clase/Tipo de Defecto | PF | NF | Total |
|---|---|---|---|
| Ordenador | 2 | 5 | 7 |
| OrdenadorSinRep | 3 | 3 | 6 |
| Total | 5 | 8 | 13 |

## 6.3. Sujetos

En las experiencias iniciales participan 21 sujetos de similares características. En la primer experiencia participan 10 y en la segunda 11. Todos ellos son estudiantes de la carrera Ingeniería en Computación de la Facultad de Ingeniería. Son estudiantes avanzados dado que todos se encuentran en cuarto o quinto año. Tienen aprobado el curso de Taller de Programación[1] en el cual se aprende Java, y el curso de Introducción a la Ingeniería de Software[2] en el cual se estudian diversas técnicas de verificación.

Consideramos que el conjunto de sujetos que participa en el experimento es homogéneo debido a su similar avance en la carrera, y a las clases niveladoras que son dictadas con el objetivo de brindar conocimientos teóricos y prácticos sobre las técnicas utilizadas.

## 6.4. Diseño del Experimento

El experimento consta de una **única unidad experimental**: el programa de ordenamiento de array. Posee además un **único factor** con **dos alternativas**: las técnicas de verificación a evaluar CS y TU. El diseño elegido es típico para experimentos de un factor con dos alternativas. El experimento es realizado en dos instancias (experiencias I y II) con una diferencia de 3 semanas entre una y otra. Como se explicó anteriormente, estas instancias son realizadas en el marco del entrenamiento de los sujetos para realizar un experimento posterior de mayor porte.

La asignación de las técnicas a los sujetos es totalmente aleatoria. El procedimiento seguido para realizar la asignación en cada experiencia inicial es descrito

[1] Curso de 3er año de la carrera Ingeniería en Computación de la Facultad de Ingeniería - Universidad de la República.

[2] Curso de 4to año de la carrera Ingeniería en Computación de la Facultad de Ingeniería - Universidad de la República.

a continuación. Para la experiencia I se anotan los nombres de los sujetos en diez papeles y se mezclan en una bolsa. Se retiran de la bolsa de a dos papeles a la vez y se colocan al azar en dos conjuntos separados, correspondiendo cada conjunto a una de las técnicas a utilizar.

En la experiencia II se sigue el mismo procedimiento con once papeles, y se decidió que el último papel sea asignado a TU. De esta forma se logra que el diseño sea lo más balanceado posible, ya que para la primer experiencia se tiene igual cantidad de sujetos para ambas técnicas y para la segunda experiencia se tiene un sujeto más para TU.

Como fue mencionado anteriormente las variables de respuesta consideradas en este experimento son el costo y la efectividad de las técnicas. La efectividad se define como la cantidad de defectos encontrados sobre la cantidad de defectos totales expresada en porcentaje. El costo es entendido como el tiempo que requiere la aplicación de la técnica, esto equivale al tiempo que lleva diseñar los casos de prueba y codificarlos en JUnit.

La hipótesis nula de efectividad, hipótesis que se quiere rechazar, plantea que las medianas de la efectividad de las técnicas son iguales. La hipótesis nula de costo plantea que las medianas de costo de las técnicas son iguales. Las hipótesis alternativas correspondientes simplemente indican que las medianas son diferentes.

Los participantes de cada experiencia verifican el programa en un único día en el mismo salón. Con esto se busca evitar el intercambio de información ya que los sujetos se encuentran bajo la supervisión de los tutores (investigadores). El hecho de compartir la ubicación física favorece además la rápida respuesta a las dudas de los estudiantes.

## 6.5. Operación

Cada experiencia se realiza en tres sesiones: una sesión inicial de aprendizaje, una sesión de capacitación, y una de ejecución individual. La figura **??** ilustra las sesiones llevadas a cabo en las experiencias.

La sesión de aprendizaje tiene como objetivo que cada sujeto aprenda JUnit. Se proporciona una especificación de un programa sencillo a todos los sujetos, y se solicita que implementen dicha especificación en Java y que desarrollen una clase JUnit que verifique su funcionalidad. Se pretende que el sujeto repase Java y estudie Junit ya que no se explica esta herramienta durante la capacitación. Esta sesión es llevada a cabo por los estudiantes en sus casas y tiene una duración de una semana. Al finalizar la misma los estudiantes entregan la clase Java y la clase JUnit con los casos de prueba codificados. Estas entregas son revisadas por las tutoras para verificar que el estudiante haya adquirido el conocimiento necesario con JUnit.

La sesión de capacitación es realizada durante un día completo. En la primer mitad del mismo se proporciona a los sujetos una clase teórica, en la cual se realiza un repaso de algunos conceptos referentes a Ingeniería de Software y se explican las técnicas a utilizar. También se presenta la guía de verificación 5.2.2 que deben seguir, y la planilla Grillo 5.2.3 donde se registra el tiempo invertido en el diseño de los casos de prueba, el tiempo invertido en encontrar cada defecto, y los defectos encontrados. Durante la segunda parte del día se llevan a cabo ejercicios prácticos en forma grupal e individual. Para lograr una

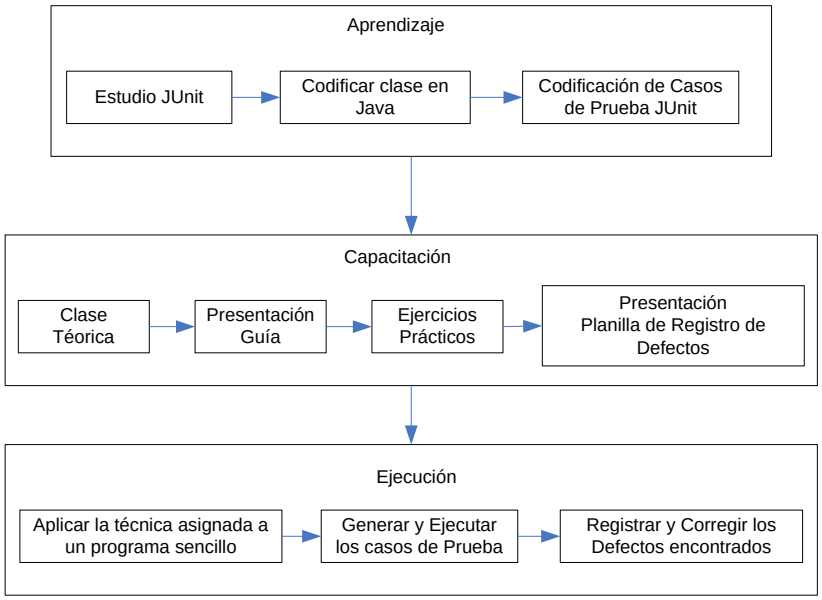



Figura 6.7: Sesiones de la fase de Operación de las Experiencias Inciales

jornada amena se realiza un almuerzo compartido al mediodía.

La clase brindada a los sujetos es preparada y dictada por las autoras de este proyecto y cuenta con la supervisión del tutor de este proyecto. Las diapositivas de las clase se presentan en 5.2.4. Los ejercicios prácticos también son preparados por los tutores seleccionando clases Java de una tarea del curso Taller de Programación. Se realiza una selección de clases Java para que resulte interesante aplicar Intra-método e Inter-método.

La sesión de ejecución individual se realiza también durante todo un día. Entre la sesión de capacitación y la de ejecución hay 7 días de diferencia en el tiempo. Durante esa semana los sujetos repasan la clase dictada y envían sus dudas a los tutores. Durante esta sesión cada sujeto aplica de forma individual la técnica que le fue asignada, generando los casos de prueba necesarios y ejecutando los mismos. Para llevar a cabo el trabajo se sigue la guía que fue proporcionada en la capacitación, registrando los tiempos y defectos tal como se indica en la misma. En esta jornada también se realiza un almuerzo compartido al igual que en la capacitación. Los estudiantes que no logran terminar su tarea en el transcurso del día cuentan con una semana de plazo para realizar la entrega. Se solicita que entreguen las clases JUnit generadas, las planillas de registro de tiempos y defectos, y las anotaciones que hayan efectuado para poder aplicar las técnicas (grafos de flujo de control, caminos identificados, casos de prueba, etc). Una vez efectuadas todas las entregas, los tutores revisan las mismas y realizan una devolución individual a cada estudiante.

La experiencia I es muy útil para adquirir información valiosa sobre la adecuación de la capacitación brindada a los estudiantes, y permite mejorarla para la experiencia II. En base al desarrollo y resultados de la experiencia I se realizan mejoras a la clase teórica que se brinda a los estudiantes y se ajustan algunos aspectos del proceso seguido.

## 6.6. Errores Cometidos por los Sujetos

Como se mencionó, al finalizar la sesión de ejecución se solicita a los sujetos que entreguen las clases JUnit generadas, las planillas de registro, y las anotaciones en papel que hayan realizado al aplicar las técnicas. Es importante destacar que no se pide a los sujetos que realicen una explicación escrita de cada paso seguido al aplicar las técnicas, sino que se solicita que entreguen el material generado de forma espontánea durante la ejecución. Decidimos esto, ya que el hecho de exigir al sujeto la generación de anotaciones impactaría en el costo asociado a la técnica. El sujeto consumiría bastante más tiempo en realizar todos los grafos y caminos, además no le permitiría aplicar la técnica de forma libre.

Las entregas son revisadas por los tutores con el fin de observar el grado de entendimiento de las técnicas y su aplicación, siendo conscientes de que no se solicitó que describan la solución paso a paso por escrito. Por ello no puede corroborarse con certeza el entendimiento de todos los aspectos de las técnicas.

Como resultado de la revisión se identifican varios errores cometidos en la aplicación de las técnicas y algunos en la utilización de JUnit. Los errores cometidos por los sujetos en las experiencias I y II pueden visualizarse en el Cuadro 6.2 y en el Cuadro 6.3 respectivamente, clasificados por sujeto. Estos errores son notificados a los sujetos correspondientes para lograr que la capacitación sea satisfactoria, buscando que el sujeto comprenda mejor la técnica.

Se realiza un análisis de los errores cometidos en ambas experiencias de forma separada. El motivo de este análisis diferenciado es para analizar si los ajustes realizados a la experiencia II impactan de alguna manera en la aplicación de las técnicas.

Para analizar y comparar los errores cometidos se realiza una clasificación de los mismos en los Cuadros 6.4 y 6.5 para las experiencias I y II respectivamente. Las filas representan los tipos de errores cometidos y las columnas corresponden a las técnicas con las que se cometen dichos errores. Cada celda contiene la cantidad de sujetos que cometen ese tipo de error (fila) con esa técnica (columna).

Puede observarse que en ambas experiencias la cantidad de errores cometidos al aplicar TU es mucho mayor que la cantidad de errores correspondiente a la aplicación de CS. Resulta evidente que la técnica TU es más difícil de aplicar, y por lo tanto, los sujetos se equivocan más.

Los errores cometidos en la aplicación de CS se deben a errores en la construcción de los CFG de los métodos a probar, y a errores en la construcción de los casos de prueba codificados en JUnit. Estos errores no pueden atribuirse directamente a la técnica.

Los errores correspondientes a la aplicación de TU son más variados, incluyendo errores en la construcción de los CFG, errores relacionados directamente a la aplicación de la técnica, y errores en los casos de prueba codificados en JUnit.

Observamos que la cantidad de errores cometidos es muy similar en ambas experiencias. Aplicando CS, se comete un error más en la experiencia II con respecto a la experiencia I. Los errores cometidos para CS no pueden ser atribuidos directamente a la técnica.

En la experiencia inicial II, la cantidad de sujetos que se equivocan aplicando TU aumenta en dos en comparación a la experiencia I.

Este análisis no nos permite obtener conclusiones sobre si los cambios realizados para la capacitación brindada en la experiencia II afectaron positivamente

Cuadro 6.2: Errores Cometidos en la Experiencia Inicial I

| Sujeto | Técnica | Error |
|---|---|---|
| S1 | CS | - Error de construcción en CFG de ordenarSinRep. |
| S2 | CS | No se detectan errores. |
| S3 | CS | No se detectan errores. |
| S4 | CS | - Error de construcción en CFG de desplazar. |
| S5 | CS | No se detectan errores. |
| S6 | TU | - En los CFG no divide las sentencias for en nodos individuales.<br>- a[j] >a[j+1] ->no identifica el p-uso de j.<br>- aux = a[j], a[j+1]=a[j], a[j]=aux ->no identifica los c-usos de j.<br>- Ordenador.ordenar(a) ->no identifica la definición de a.<br>- a[i] == a[i+1] ->no identifica el p-uso de i.<br>- desplazar(a,i+1) ->no identifica la definición de a.<br>- a[j]=a[j+1] ->no identifica el c-uso de j.<br>- En los casos Junit indica el array entero como resultado esperado, la composición del array entero no puede ser derivada de la especificación. |
| S7 | TU | - Ordenador.ordenar(a) ->no identifica la definición de a.<br>- desplazar(a,i+1) ->no identifica la definición de a.<br>- En Inter no considera las definiciones de a en el método invocado por el método a probar. |
| S8 | TU | - a[j] = aux ->no identifica el c-uso de j.<br>- i <a.lenght - 1 - cantEliminados ->no identifica el p-uso de i.<br>- Para nodos donde existen definición y uso de una misma variable, no busca un camino libre definición que lleve desde la definición al uso. |
| S9 | TU | - j++ ->no identifica el c-uso ni la definición de j (error de distracción).<br>- a[i]==a[i+1] ->registra c-uso de i en lugar de p-uso.<br>- Considera la variable cantEliminados en Inter. |
| S10 | TU | - No sigue enfoque bottom-up.<br>- Ordenador.ordenar(a) ->no identifica la definición de a.<br>- desplazar(a,i+1) ->no identifica la definición de a.<br>- Error de construcción en CFG de ordenar.<br>- No busca caminos CLD desde las definiciones pertenecientes a un método invocado por el método a testear. |

a la aplicación de las técnicas. Al parecer las mejoras en la capacitación no ayudaron significativamente a reducir la cantidad de errores. Como son pocos los sujetos aplicando las técnicas, un número mayor de estos en las experiencias sería necesario para poder determinar si la capacitación de la experiencia II

Cuadro 6.3: Errores Cometidos en la Experiencia Inicial II

| Sujeto | Técnica | Error |
|---|---|---|
| S11 | CS | No se detectan errores. |
| S12 | CS | - Error de construcción en CFG de ordenar.<br>- En los casos de prueba Junit de ordenadorSinRep no verifica el array devuelto, solo la cantidad de repetidos. |
| S13 | CS | No se detectan errores. |
| S14 | CS | No se detectan errores. |
| S15 | CS | - En los casos de prueba JUnit de ordenadorSinRep no verifica el array devuelto, solo la cantidad de repetidos. |
| S16 | TU | - a[j]>a[j+1] ->no identifica el p-uso de j.<br>- aux = a[j] ->no identifica c-uso de j.<br>- a[j+1] = a[j] ->no identifica c-uso de j.<br>- a[j] = aux ->no identifica c-uso de j.<br>- Ordenador.ordenar(a) ->no identifica la definición de a.<br>- No aplica Intra al método privado desplazar.<br>- a[i] == a[i+1] ->no identifica p-uso de i.<br>- desplazar(a,i+1) ->no identifica definición de a. |
| S17 | TU | - En los casos JUnit indica el array entero como resultado esperado, sin embargo la composición del array entero no puede ser derivada de la especificación. |
| S18 | TU | No se detectaron errores. |
| S19 | TU | - Trabajo innecesario durante el diseño. |
| S20 | TU | - a[j] >a[j+1] ->identifica c-uso de j, siendo un p-uso de j.<br>2. a[i] == a[i+1] ->no identifica p-uso de i. |
| S21 | TU | - a[j]>a[j+1] ->no identifica el p-uso de j.<br>- aux = a[j] ->no identifica c-uso de j.<br>- a[j+1] = a[j] ->no identifica c-uso de j.<br>- a[j] = aux ->no identifica c-uso de j.<br>- No aplica Intra al método privado desplazar.<br>- a[i] == a[i+1] ->no identifica p-uso de i.<br>- desplazar(a,i+1) ->no identifica definición de a (pero busca CLD partiendo de esta def), no c-uso de i (Intra).<br>- Escribe nodo Ordenador.R en lugar de R.<br>- En los casos de prueba JUnit de ordenadorSinRep no verifica el array devuelto, solo la cantidad de repetidos. |

realmente logra o no un mejor entendimiento de la técnica.

Cuadro 6.4: Clasificación de defectos - Experiencia I

| Errores | Cantidad de Sujetos | |
|---|---|---|
| | CS | TU |
| Error de construcción en CFGs. | 2 | 2 |
| Omitir la definición del array a en las sentencias Ordenador.ordenar(a) y desplazar(a, i+1). | - | 3 |
| Error en la identificación de usos de las variables utilizadas en expresiones como índices de un array. | - | 2 |
| Otros errores en pruebas a nivel de Intra-método. | - | - |
| Otros errores en pruebas a nivel de Inter-método. | - | 3 |
| Error en la construcción de los casos de prueba implementados en Junit. | 0 | 1 |
| Otros errores. | 0 | 1 |

Cuadro 6.5: Clasificación de defectos - Experiencia II

| Errores | Cantidad de Sujetos | |
|---|---|---|
| | CS | TU |
| Error de construcción en CFGs. | 1 | 1 |
| Omitir la definición del array a en las sentencias Ordenador.ordenar(a) y desplazar(a, i+1). | - | 2 |
| Error en la identificación de usos de las variables utilizadas en expresiones como índices de un array. | - | 3 |
| Otros errores en pruebas a nivel de Intra-método. | - | 2 |
| Otros error en pruebas a nivel de Inter-método. | - | - |
| Error en la construcción de los casos de prueba implementados en Junit. | 2 | 2 |
| Otros errores. | 0 | 0 |

## 6.7. Resultados

Lo primero a resolver antes de analizar los resultados es si se pueden considerar las dos experiencias como un único experimento. Es decir, si se pueden analizar los resultados en conjunto, o se deben realizar los análisis de resultados de forma independiente para cada una de las experiencias. La diferencia entre las experiencias I y II son los cambios realizados en la sesión de capacitación. En la sección 6.6 se indica que estas diferencias no impactaron significativamente en los defectos encontrados en ambas experiencias. Entendemos que los cambios realizados son menores y que no afectaron el comportamiento de los sujetos. Por lo tanto, se realiza el análisis de los datos obtenidos en ambas experiencias de forma conjunta, considerando ambas experiencias como un único experimento.

En el Cuadro 6.6 se presentan los defectos detectados por cada uno de los sujetos. Los 10 primeros sujetos participan de la experiencia I mientas que los restantes participan de la experiencia II. Las filas representan a los sujetos y las columnas representan los distintos defectos del programa. El valor de una celda corresponde al tiempo en minutos que le llevó al sujeto detectar el defecto una vez que observó la falla mediante la ejecución de un caso de prueba. Los casos en que el sujeto no detectó el defecto aparecen con el siguiente símbolo "-".

Se determina también la efectividad y el costo de la aplicación de la técnica por cada sujeto. Se define la efectividad de la técnica como la cantidad de defectos encontrados sobre la cantidad de defectos totales expresada en porcentaje. El costo de la técnica se define como el tiempo que insume diseñar y codificar en JUnit los casos de prueba. Para análisis posteriores se usará el tiempo que requiere encontrar los defectos. En el Cuadro 6.7 se muestran los valores de efectividad y costo por sujeto. La penúltima columna del cuadro indica la efectividad (en porcentaje) de cada sujeto al aplicar la técnica sobre el programa. La última columna es el tiempo (costo) utilizado por el sujeto en diseñar y codificar los casos de prueba.

Cuadro 6.6: Detección de Defectos

| | | Defectos | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sujeto | Técnica | A | B | C | D | E | F | G | a | b | c | d | e | f |
| S1 | CS | - | 0 | - | - | - | - | 0 | - | 12 | - | 13 | - | - |
| S2 | CS | - | 10 | - | - | - | - | 0 | - | - | - | 0 | - | - |
| S3 | CS | - | 0 | - | - | - | - | 0 | - | 5 | - | 5 | - | - |
| S4 | CS | - | 1 | - | - | - | - | - | - | - | - | - | - | - |
| S5 | CS | 0 | 4 | - | - | - | - | 0 | - | - | - | - | - | - |
| S6 | TU | - | 0 | - | - | - | - | 0 | - | 20 | - | 10 | - | - |
| S7 | TU | - | 0 | - | - | - | - | 0 | 0 | - | - | 2 | - | - |
| S8 | TU | - | 0 | - | - | - | - | 0 | - | 10 | - | 6 | - | - |
| S9 | TU | - | 2 | - | - | - | - | 0 | - | 3 | - | 5 | - | - |
| S10 | TU | 1 | 4 | - | - | - | - | 0 | 1 | - | - | 8 | - | - |
| S11 | CS | - | 4 | - | - | - | - | 0 | - | - | - | 35 | - | - |
| S12 | CS | - | 3 | - | - | - | - | 0 | - | 2 | - | 32 | - | - |
| S13 | CS | 0 | 0 | - | - | - | - | 0 | 0 | 0 | - | - | - | - |
| S14 | CS | 0 | 0 | - | - | - | - | 0 | - | 9 | - | 3 | - | - |
| S15 | CS | 0 | 5 | - | - | - | - | 0 | 0 | 30 | - | 90 | - | - |
| S16 | TU | - | 10 | - | - | - | - | 0 | - | - | - | 5 | - | - |
| S17 | TU | - | 10 | - | - | - | - | 0 | - | 54 | - | 7 | - | - |
| S18 | TU | 1 | 4 | - | - | - | - | 0 | 1 | 7 | - | 16 | - | - |
| S19 | TU | - | 0 | - | - | - | - | 0 | - | 0 | - | 0 | - | - |
| S20 | TU | 0 | 4 | - | - | - | - | 0 | - | 25 | - | 3 | - | - |
| S21 | TU | 1 | 4 | - | - | - | - | 0 | 1 | 5 | - | 25 | - | - |

### 6.7.1. Efectividad de las Técnicas

En el Cuadro 6.8 se presentan la media, mediana y la desviación estándar de cada técnica con respecto a la Efectividad.

Algunas observaciones preliminares acerca de la efectividad que se obtienen del Cuadro 6.8 son las siguientes:

- En promedio (valor de la media), la técnica TU con valor de 34,3 %, resulta ser más efectiva que CS, de la cual se observa un valor de efectividad promedio de 29,2 %.

- La desviación estándar de TU en relación a la efectividad es menor que

Cuadro 6.7: Efectividad y Costo por Sujeto

| Sujeto | Técnica | Efec. % | Costo |
|---|---|---|---|
| S1 | CS | 30,77 | 103 |
| S2 | CS | 23,08 | 165 |
| S3 | CS | 30,77 | 180 |
| S4 | CS | 7,69 | 174 |
| S5 | CS | 23,08 | 66 |
| S6 | TU | 30,77 | 280 |
| S7 | TU | 30,77 | 320 |
| S8 | TU | 30,77 | 500 |
| S9 | TU | 30,77 | 207 |
| S10 | TU | 38,46 | 385 |
| S11 | CS | 23,08 | 75 |
| S12 | CS | 30,77 | 113 |
| S13 | CS | 38,46 | 240 |
| S14 | CS | 38,46 | 90 |
| S15 | CS | 46,15 | 120 |
| S16 | TU | 23,08 | 133 |
| S17 | TU | 30,77 | 514 |
| S18 | TU | 46,15 | 237 |
| S19 | TU | 30,77 | 632 |
| S20 | TU | 38,46 | 215 |
| S21 | TU | 46,15 | 185 |

Cuadro 6.8: Media, Mediana y Desviación Estándar de las técnicas

| | Media | Mediana | Desviación Estándar |
|---|---|---|---|
| CS | 29,2 % | 30,77 % | 10,8 % |
| TU | 34,3 % | 30,77 % | 7,2 % |

CS. Se observa que la técnica TU se desvía en promedio de su media en 7,2, mientras que CS en 10,8.

En la sección 6.6 de Errores Cometidos por los sujetos se observa que la técnica de CS fue mejor aplicada que TU, en el sentido que se cometieron menos errores relacionados a su aplicación. Dados los tipos de errores detectados, se puede suponer que de haberse aplicado TU de mejor forma se habría obtenido una mayor efectividad para dicha técnica. De todas formas esto no es posible afirmarlo.

Se aplican dos test no paramétricos para la hipótesis nula que expresa que las medianas de efectividad de ambas técnicas son iguales:

$H_0 : \mu_{CS} = \mu_{TU}$

$H_1 : \mu_{CS} <> \mu_{TU}$

Los test utilizados son el de Mann-Whitney y el de Kruskal Wallis. Ninguno de los test rechaza $H_0$ con $\alpha \leq 0,1$. Entonces, las diferencias de efectividad que se aprecian no son significativas estadísticamente. Se necesitan mas observaciones así como trabajar con distintos programas para obtener resultados más significativos.

No se realiza ningún análisis de efectividad por tipo de defecto. En este experimento se tienen pocos defectos y cualquier análisis en este sentido no sería válido estadísticamente.

Mencionamos anteriormente un experimento que se realizó con el mismo programa que este experimento [21] pero con otras técnicas de verificación. En dicho experimento se estudian algunos aspectos cualitativos de la efectividad. Estos aspectos se pueden analizar para este experimento e incluso comparar con el anterior.

Una de las observaciones del experimento mencionado es que **los defectos PF se encuentran más fácilmente que los NF.** En el Cuadro 6.9 se presenta qué tan efectivas son las técnicas respecto a cada defecto en este experimento. Esto se calcula dividiendo la cantidad de sujetos que encontró el defecto entre la cantidad total de sujetos. Se muestran los datos generales de las dos experiencias discriminados por técnica y en total.

Cuadro 6.9: Porcentaje de detección de defectos

| Técnica | Defectos | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | a | b | c | d | e | f |
| CS | 40 | 100 | 0 | 0 | 0 | 0 | 90 | 20 | 60 | 0 | 70 | 0 | 0 |
| TU | 36 | 100 | 0 | 0 | 0 | 0 | 1 | 36 | 72 | 0 | 1 | 0 | 0 |
| Promedio | 38 | 100 | 0 | 0 | 0 | 0 | 95 | 28 | 66 | 0 | 86 | 0 | 0 |

Los defectos PF son A, B, a, b y d. Los defectos que son encontrados durante este experimento son esos y además el defecto G. El resto de los defectos no se detecta. Entonces, se puede concluir lo mismo que en el experimento anterior.

Las técnicas dinámicas buscan defectos mediante la provocación de fallas. Por lo tanto, es bastante esperada esta primer conclusión. Sin embargo, los sujetos, luego de provocada una falla, revisan el código para buscar el defecto. También se lleva a cabo una revisión del código durante la instancia de creación de los casos de prueba. Lo que muestra este experimento es que, cuando se revisa el código con el objetivo de buscar un defecto que provocó una falla o generar los casos de prueba se pasan por alto otros defectos.

Otro hallazgo del experimento anterior es que **los defectos de rendimiento (*performace*) no son fáciles de detectar**. El defecto f es el único defecto de performance en el programa. En el experimento anterior fue el único defecto que no fue detectado. En este experimento 7 defectos de los 13 no fueron detectados. Por lo tanto, no podemos concluir que los defectos de rendimiento sean más difíciles de detectar que otros defectos que no provocan fallas.

Al igual que en el experimento anterior se concluye que la efectividad de cada técnica es baja. Los promedios de efectividad detectados en ese experimento son: Insp 31 %, CCCM 17 %, CE 27 %, TD 31 %. En este experimento los promedios fueron presentados en el Cuadro 6.8. La técnica TU presenta la mayor efectividad considerando las 6 técnicas estudiadas.

### 6.7.2. Costo de las Técnicas

El costo asociado a las pruebas realizadas por cada sujeto se presentó en el Cuadro 6.7. El Cuadro 6.10 presenta los costos en minutos para las dos clases Java del programa y el total discriminado por sujeto.

Cuadro 6.10: Costos de las Técnicas por Sujeto

| Sujeto | Técnica | Tiempo Ordenador | Tiempo Ordenador-SinRep | Tiempo Total |
|---|---|---|---|---|
| S1 | CS | 57 | 46 | 103 |
| S2 | CS | 75 | 90 | 165 |
| S3 | CS | 60 | 120 | 180 |
| S4 | CS | 80 | 94 | 174 |
| S5 | CS | 36 | 30 | 66 |
| S6 | TU | 145 | 135 | 280 |
| S7 | TU | 170 | 150 | 320 |
| S8 | TU | 190 | 310 | 500 |
| S9 | TU | 81 | 126 | 207 |
| S10 | TU | 145 | 240 | 385 |
| S11 | CS | 45 | 30 | 75 |
| S12 | CS | 62 | 51 | 113 |
| S13 | CS | 120 | 120 | 240 |
| S14 | CS | 60 | 30 | 90 |
| S15 | CS | 90 | 30 | 120 |
| S16 | TU | 37 | 96 | 133 |
| S17 | TU | 178 | 336 | 514 |
| S18 | TU | 83 | 154 | 237 |
| S19 | TU | 242 | 390 | 632 |
| S20 | TU | 135 | 80 | 215 |
| S21 | TU | 57 | 128 | 185 |

El costo promedio de aplicar la técnica TU es mayor a aplicar CS. El promedio de tiempo invertido para cubrimiento de sentencias fue de 133 minutos mientras que para todos los usos fue de 328 minutos.

El Cuadro 6.11 presenta la media, mediana y la desviación estándar de cada técnica con respecto al Costo.

Cuadro 6.11: Media, Mediana y Desviación Estándar de las técnicas

| | Media | Mediana | Desviación Estándar |
|---|---|---|---|
| CS | 132,6 | 116,5 | 52,4 |
| TU | 328 | 280 | 152,6 |

Se obtienen del Cuadro 6.11 las siguientes observaciones acerca de costo de las técnicas CS y TU:

- En promedio (valor de la media), la técnica TU resulta ser mucho más costosa que CS, con valores 328 y 132,6 para TU y CS respectivamente.
- La desviación estándar de TU en relación al costo es mucho mayor que CS. Se observa que la técnica TU se desvía en promedio de su media en 152,6, mientras que CS en 52,4.

Cómo ya mencionamos, se cometieron más errores por parte de los sujetos al aplicar TU que CS. Probablemente la ejecución correcta de TU implicaría un

costo aún mayor del observado en el cuadro 6.11. Consideramos esto debido a que en algunos casos no se identificaron usos de variables, lo que podría implicar la omisión de casos de prueba necesarios para cumplir con la técnica.

Nuevamente se usan los test no paramétricos Mann-Whitney y Kruskal Wallis. La hipótesis nula es que las medianas de costo de las técnicas son iguales. Ambos test rechazan dicha hipótesis con $\alpha \leq 0,1$. Como era de esperar, podemos concluir que existe suficiente evidencia estadística como para afirmar que TU es más costosa que CS.

De todas formas se deben realizar más experimentos. Es muy importante realizar experimentos con programas más complejos y tener más sujetos para tener más y mejores observaciones. El experimento detallado en el capítulo 7, se realiza sobre un programa mucho más complejo que el de este experimento (experiencias iniciales). Sin embargo, la cantidad de sujetos no se pudo aumentar.

En el experimento con 5 técnicas y con este mismo programa se tuvieron solamente entre 3 y 4 sujetos por técnica. Para una misma técnica existió una alta variabilidad en el costo. Al ser pocas observaciones y con alta variabilidad preferimos no realizar comparaciones de costo entre este y el otro experimento.

### 6.7.3. Costo de Detección de Defectos

Es interesante conocer qué defectos son más costosos de encontrar. En este experimento 6 defectos fueron detectados por los sujetos. El Cuadro 6.12 presenta para cada uno de esos defectos el costo promedio en detectarlos y la desviación estándar. Este costo es la suma de los tiempos de detección de cada sujeto que detectó el defecto dividido la cantidad de sujetos que detectó el defecto. El costo está expresado en minutos.

Cuadro 6.12: Costo Promedio en Detectar Cada Defecto

| Defectos | A | B | G | a | b | d |
|---|---|---|---|---|---|---|
| Costo (min) | 0,38 | 3,1 | 0 | 0,5 | 13 | 14,72 |
| Desv. est. | 0,52 | 3,42 | 0 | 0,55 | 14,97 | 21,48 |

Si bien existen diferencias sustanciales en los promedios de detección de los defectos, las desviaciones estándares asociadas son tan grandes o más que los promedios. Los defectos con los que se cuenta son muy pocos y no tiene mayor sentido clasificarlos para conocer qué tipo de defectos es más costoso de encontrar. Se necesita mayor cantidad de defectos, así como tenerlos agrupados por tipos de defecto, para poder discutir acerca del costo de detección de cada tipo.

También puede ser interesante discutir acerca de si el costo de detección de defectos varía según la técnica que fue aplicada. Si bien la etapa de detección de defectos es igual para ambas técnicas, pueden existir diferencias en el costo de detección. Para este análisis aún se tienen menos muestras por lo que preferimos no realizarlo.

## 6.8. Conclusiones

A lo largo del capítulo se presentan las experiencias I y II con dos visiones diferentes. Por un lado, se presentan desde el punto de vista de la capacitación brindada a los sujetos para realizar el experimento formal, y por otro, se analizan como un experimento en sí mismo.

Desde el punto de vista de la capacitación, con estas experiencias se logra brindar la inducción necesaria a los sujetos para poder realizar el experimento formal. Se brindan clases teórico-prácticas acerca de la aplicación de las técnicas y del uso de los materiales de soporte. El entrenamiento consiste en la aplicación de las técnicas CS y TU por parte de los sujetos. Durante las experiencias se realizan correcciones a los sujetos en lo que respecta a desviaciones encontradas en la aplicación de las técnicas con el objetivo de minimizar futuros errores. También se realizan correcciones sobre el registro de los defectos detectados en la planilla Grillo.

Por otro lado, se presentan las experiencias iniciales como un experimento en sí mismo, con el propósito de conocer el comportamiento de las técnicas de pruebas CS y TU. El experimento consiste en 21 sujetos realizando pruebas sobre un programa sencillo, 10 sujetos aplican CS y 11 aplican TU. Se analizan tanto la efectividad como el costo de las técnicas. Los resultados indican que estadísticamente no se puede concluir que una técnica es más efectiva que la otra. Sin embargo, parece ser más efectiva TU. Más experimentos deben ser realizados para concluir sobre este aspecto. En lo que respecta al costo, se presenta evidencia estadística que indica que la técnica TU es mas costosa que CS, como era esperado. Sin embargo, el experimento se realiza utilizando un único y simple programa. Más experimentos, sobre todo utilizando diferentes programas, son necesarios para confirmar esta afirmación.

# Capítulo 7

# Experimento Formal

En este capítulo se presenta el experimento formal que busca comparar el comportamiento de las técnicas de verificación Cubrimiento de Sentencias (CS) y Todos los Usos (TU). Se busca analizar las técnicas con el propósito de conocer su efectividad y costo a nivel unitario, en el contexto de un experimento controlado llevado a cabo por estudiantes de la Carrera de Ingeniería en Computación de la Facultad de Ingeniería - Universidad de la República.

El resto del capítulo se estructura de la siguiente forma. El programa utilizado se presenta en la sección 7.1 y los defectos contenidos en el mismo se describen en la sección 7.2. La sección 7.3 presenta los sujetos que participan del experimento. El diseño del experimento se explica en la sección 7.4 y su operación en la 7.5. El análisis y evaluación de los datos se presentan en la sección 7.6. En la sección 7.7 se compara el experimento formal con los experimentos formales estudiados. La sección 7.8 presenta parte de la encuesta realizada a los sujetos al finalizar el experimento. Por último se exponen las conclusiones en la sección 7.9.

## 7.1. Programa

El programa utilizado fue construido especialmente para el Experimento Año 2008 y es denominado **Contabilidad**. Fue construido por un estudiante de 4to año de la carrera de Ingeniería en Computación, a quien se le solicitó entregar el programa compilado sin realizar ningún tipo de verificación. Los únicos defectos que se permitieron corregir fueron los que impedían la compilación del programa.

Se dispone entonces de un programa cuyas faltas no son inyectadas en el código por los investigadores, sino que son las cometidas al construir el mismo. Consideramos que disponer de un programa con esta característica le da más realismo al experimento.

El estudiante que construye el programa genera la documentación del mismo utilizando Javadoc. Entrega manuales de instalación, configuración y un manual de uso que contempla todas las funcionalidades del programa.

El propósito general del programa Contabilidad es la liquidación de sueldos para funcionarios docentes y no docentes de una facultad ficticia. El sistema permite el mantenimiento de cargos y funcionarios. Ofrece la funcionalidad de reasignación de cargos, no pudiendo tener un funcionario más de un cargo (sea

docente o no docente). Permite efectuar aumentos de sueldos de distintas formas. Posee la funcionalidad de generar liquidaciones, en donde se efectúa la liquidación de todos los funcionarios del sistema creando sus respectivos recibos de sueldo. Brinda además la posibilidad de generar la liquidación para un funcionario en particular, en donde se crea el recibo de sueldo sólo para el funcionario seleccionado.

El programa está desarrollado en Java y tiene un tamaño aproximado de 1820 LOCs (sin comentarios) distribuidas en 14 clases. Está compuesto de 5 DataTypes, una clase que implementa la persistencia de los datos, 5 clases que contienen la lógica, dos interfaces, y una clase main que implementa la interfaz con el usuario. En el Cuadro 7.1 se listan las clases junto con la cantidad de líneas de código correspondientes.

El programa cuenta con una pequeña base de datos gestionada con el manejador HSQLDB para dar soporte a sus funcionalidades. Dicha base de datos está compuesta por 8 tablas. Se brinda a los sujetos un script para la creación de las tablas y carga de datos básicos.

Cuadro 7.1: Cantidad de líneas de código por clase

| Clase | LOCs (sin comentarios) |
|---|---|
| TiposDescuento | 22 |
| TiposCargo | 25 |
| Franjas | 49 |
| MaxMinCodigos | 25 |
| Descuento | 40 |
| ReciboSueldo | 114 |
| DescuentoRecibo | 35 |
| Funcionario | 137 |
| Cargo | 70 |
| Persistencia | 342 |
| IPersistencia | 28 |
| Lógica | 254 |
| ILógica | 22 |
| Main | 657 |

Se utiliza un enfoque *bottom-up* para realizar el testing de las clases. Las dependencias entre las clases se obtienen del Diagrama de Clases y de los pseudo-atributos pertenecientes a cada clase. En base a este análisis se realiza el Cuadro 7.2 que indica el orden en el que deben verificarse las clases.

## 7.2. Defectos del Programa

Los defectos del programa no se conocen en su totalidad debido a que no son defectos inyectados, sino que corresponden a los cometidos en su desarrollo. Se dispone de una gran cantidad de defectos detectados durante el Experimento Año 2008, y durante la ejecución de este experimento. Consideramos que con la unión de los defectos detectados en ambos experimentos se está muy cerca de contar con la cantidad de defectos totales del programa. A esta cantidad de defectos la denominamos estimado de la cantidad de defectos.

Cuadro 7.2: Enfoque *bottom-up* para las clases del programa

| Clase | Orden de Testeo |
|---|---|
| TiposDescuento | 1ra |
| TiposCargo | 1ra |
| Franjas | 1ra |
| MaxMinCodigos | 1ra |
| DescuentoRecibo | 1ra |
| Descuento | 2da |
| Cargo | 2da |
| ReciboSueldo | 3ra |
| Funcionario | 4ta |
| Persistencia | 5ta |
| IPersistencia | 5ta |
| Lógica | 6ta |
| ILógica | 6ta |
| Main | 7ta |

## 7.3. Sujetos

Participan un subconjunto de los sujetos que participan en las experiencias iniciales, 4 estudiantes de la experiencia I y 10 de la experiencia II. Los sujetos forman parte de este experimento en el marco de una materia de Facultad con una cierta cantidad de créditos asignados. Desconocen que están participando en un experimento, por lo cual su motivación está asociada solamente a la aprobación del curso. De esta forma se evitan desvíos que pueden producirse en el trabajo de los sujetos por sentirse observados.

Consideramos que el conjunto de sujetos que participan en el experimento es homogéneo debido a su similar avance en la carrera y su participación en las experiencias iniciales, dictadas con el objetivo de brindar conocimientos teóricos y prácticos sobre la aplicación de las técnicas y el uso de los materiales de soporte.

## 7.4. Diseño del Experimento

El diseño de este experimento es de un factor (técnica) con dos alternativas (CS y TU) presentado en el capítulo 2. Consta de una única unidad experimental: el programa Contabilidad. Las variables de respuesta consideradas en este experimento son la efectividad y el costo de las técnicas. La efectividad se define como la cantidad de defectos encontrados sobre la cantidad de defectos totales expresada en porcentaje. El costo es entendido como el tiempo que requiere la aplicación de la técnica, esto equivale al tiempo que lleva diseñar los casos de prueba y codificarlos en JUnit.

La ejecución del experimento se realiza en una única instancia con una duración de 8 semanas. Los 14 sujetos prueban el mismo programa, 6 de ellos aplican CS y 8 aplican TU.

Se mantiene la asignación aleatoria de técnicas a sujetos realizada en las experiencias iniciales. Consideramos importante mantener dicha asignación para

que los sujetos apliquen la misma técnica con la que practicaron durante el entrenamiento. Si bien son más los sujetos asignados a TU decidimos mantener esta asignación.

## 7.5. Operación

La primer etapa de la fase de operación es proveer a los sujetos de la capacitación necesaria en la aplicación de las técnicas y el uso de los materiales de soporte para la ejecución de este experimento. La capacitación consiste en brindar a los sujetos clases teórico-prácticas explicando las técnicas, la Guía a seguir y la planilla de registro de defectos. Además, cada sujeto realiza una sesión de ejecución individual, que consiste en la aplicación de las técnicas a un simple programa escrito en Java. Esta capacitación corresponde a las experiencias iniciales explicadas en detalle en el capítulo 6.

El experimento se realiza en 8 sesiones de una semana de duración cada una. Cada sesión tiene asignada una cierta cantidad de clases para ser probadas por los sujetos en el correr de dicha semana. La elección de las clases a verificar por semana se realiza en base al enfoque *bottom-up* y a la complejidad de las mismas. La complejidad se estima en base a juicio de expertos, teniendo en cuenta la cantidad de líneas de código de la clase, cantidad de métodos y complejidad de los mismos.

La ejecución del experimento por parte de los sujetos se realiza a distancia. Al comienzo de cada semana se envía a los sujetos por correo electrónico las clases a verificar. Durante la semana los sujetos reportan los defectos encontrados a sus tutoras (investigadores) solicitando corrección. Las tutoras envían para cada defecto encontrado la corrección del mismo. Los sujetos no están autorizados a realizar las correcciones por sí solos, sino que siempre deben solicitarlas. Se busca con esto que todos los sujetos tengan la misma corrección para los mismos defectos. Los defectos reportados por cada sujeto quedan registrados en su planilla Grillo con toda la información solicitada en la misma, de forma de identificar el defecto en el programa.

Al comienzo del experimento se acuerda con cada sujeto el día de la semana en el que realiza sus entregas. Las mismas se componen de los casos de prueba generados y la planilla de registro de defectos.

Las entregas son revisadas para validar si la planilla de defectos está completa, es decir, si se registran detalladamente los defectos y los tiempos requeridos. En caso contrario, se solicita al sujeto que realice las correcciones y vuelva a entregar. Además, se controla que los casos de prueba entregados codificados en JUnit no fallen, lo que significa que todos los defectos detectados (que causan falla) han sido corregidos.

Todas las semanas los sujetos entregan las notas realizadas para la aplicación de las técnicas (grafos de flujo de control, caminos identificados, casos de prueba, etc.). Esta actividad no es obligatoria, se entregan las notas en caso que el estudiante haya necesitado realizarlas. Se decide esto porque se prentende que el sujeto aplique la técnica libremente, realizando o no diagramas de flujos o anotaciones especiales. Las notas son revisadas por las tutoras para evaluar el grado de apego a la técnica por parte del sujeto.

En varias ocasiones se realiza una devolución a los sujetos sobre su trabajo, en el transcurso de la semana siguiente a la entrega semanal. La entrega se revisa

y se envía un mail al estudiante comentando desviaciones encontradas y buenas prácticas, según sea el caso. Estas devoluciones intentan ser constructivas, evitando desviaciones en la aplicación de las técnicas y motivando al sujeto.

El sujeto puede adelantar semanas de testing, pero no atrasarse. Si bien se establece un día a la semana para que el sujeto realice su entrega, puede que este terminé antes de lo acordado. En ese caso tiene la posibilidad de entregar lo correspondiente a la semana en curso y solicitar las clases asignadas a la semana siguiente, de forma de adelantar trabajo.

Con los defectos detectados por los estudiantes se completa una planilla que registra los defectos detectados por clase. La misma se construye considerando la planilla de defectos resultante del Experimento Año 2008, de forma de no registrar dos veces un mismo defecto. El objetivo de la planilla creada es documentar los defectos totales del programa que han sido detectados, sin importar en cual experimento fueron reportados.

## 7.6. Análisis y Evaluación de Datos

En esta sección se realiza el análisis de los datos recolectados durante la operación del experimento. El objetivo es comparar la efectividad y el costo en la aplicación de las técnicas CS y TU.

### Efectividad de las Técnicas

Para comparar la efectividad de las técnicas se utiliza la información de la cantidad de defectos detectados por los sujetos en la ejecución del experimento.

En los Cuadros 7.3 y 7.4 se presenta la cantidad de defectos detectados por cada sujeto discriminando por clase, para la aplicación de CS y TU respectivamente. Las filas representan las clases del programa y las columnas representan los sujetos que aplican la técnica. Los sujetos de $S_1$ a $S_6$ se corresponden a los 6 sujetos que aplican CS, y los de $S_7$ a $S_{14}$ corresponden a los 8 sujetos que aplican TU. El valor de una celda representa la cantidad de defectos que detectó el sujeto en la clase de la fila correspondiente.

Cuadro 7.3: Cantidad de defectos detectados por clase para CS

| Clase | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ |
|---|---|---|---|---|---|---|
| TiposDescuento | 0 | 0 | 0 | 0 | 0 | 0 |
| TiposCargo | 0 | 0 | 0 | 0 | 0 | 0 |
| Franjas | 3 | 0 | 0 | 0 | 3 | 0 |
| MaxMinCodigos | 1 | 0 | 0 | 0 | 1 | 0 |
| Descuento | 0 | 0 | 0 | 0 | 0 | 0 |
| ReciboSueldo | 3 | 1 | 0 | 3 | 3 | 1 |
| DescuentoRecibo | 0 | 0 | 0 | 0 | 0 | 0 |
| Funcionario | 0 | 0 | 0 | 0 | 1 | 0 |
| Cargo | 0 | 0 | 0 | 0 | 0 | 0 |
| Persistencia | 36 | 12 | 4 | 4 | 13 | 4 |
| Logica | 14 | 7 | 5 | 2 | 5 | 5 |
| Main | 22 | 8 | 1 | 3 | 3 | 3 |

Cuadro 7.4: Cantidad de defectos detectados por clase para TU

| Clase | $S_7$ | $S_8$ | $S_9$ | $S_{10}$ | $S_{11}$ | $S_{12}$ | $S_{13}$ | $S_{14}$ |
|---|---|---|---|---|---|---|---|---|
| TiposDescuento | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TiposCargo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Franjas | 8 | 0 | 0 | 0 | 0 | 2 | 2 | 0 |
| MaxMinCodigos | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Descuento | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ReciboSueldo | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| DescuentoRecibo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Funcionario | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Cargo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Persistencia | 14 | 5 | 5 | 15 | 8 | 8 | 8 | 1 |
| Logica | 22 | 2 | 2 | 10 | 2 | 16 | 10 | 4 |
| Main | 16 | 0 | 3 | 3 | 0 | 18 | 18 | 3 |

A partir de la cantidad de defectos detectados y el estimado de la cantidad de defectos totales del programa se calcula la efectividad de las técnicas por sujeto. La misma es calculada a nivel del programa completo y no a nivel de las clases Java.

Se define la efectividad de la técnica como la cantidad total de defectos encontrados sobre la cantidad estimada de defectos totales, expresada en porcentaje, como se ilustra en la siguiente fórmula:

$$Efectividad = (\frac{\#TotalDefectosDetectados}{\#EstimadoDefectosTotales}) * 100$$

Como se mencionó en la sección 7.2 la cantidad de defectos totales es estimada como la sumatoria de los defectos detectados durante el Experimento Año 2008 y los detectados durante este experimento, resultando en 187 defectos en todo el programa.

En el cuadro 7.5 se presenta el total de defectos detectado por cada sujeto, y la efectividad de la técnica para cada sujeto.

Primero se interpretan los datos aplicando estadística descriptiva. Los conceptos de estadística descriptiva utilizados son presentados en el capítulo 2. Se calcula la media y la mediana para medir la tendencia central de los datos. Como medida de dispersión se utiliza la desviación estándar. Los valores calculados de estas medidas para cada técnica se presentan en el Cuadro 7.6.

Se observa que para ambas técnicas algunos datos distan significativamente de la media, los cuales pueden afectar los resultados de los test estadísticos. Estas variaciones entre los valores pueden atribuirse a las diferencias en las características de los sujetos, por ejemplo, sus habilidades, motivación, y dedicación.

Notamos durante las semanas de seguimiento que algunos sujetos fueron mucho más detallistas al realizar el testing que otros. Algunos diseñaron casos de prueba más robustos, buscando además de cumplir con la técnica probar otros casos basados en su intuición. Además, realizaron una mayor revisión del código, la cual puede darse durante el diseño de los casos de prueba, y también mientras se buscan los defectos que provocaron que los casos de prueba fallen. En cambio, otros sujetos se limitaron al diseño de casos de prueba de forma de cumplir con la técnica aplicada, y no fueron tan detallistas al leer el código.

Cuadro 7.5: Efectividad por sujeto

| Sujeto | Técnica | # Tot. Defectos Detec. | Efectividad |
|---|---|---|---|
| $S_1$ | CS | 79 | 42 % |
| $S_2$ | CS | 28 | 15 % |
| $S_3$ | CS | 10 | 5 % |
| $S_4$ | CS | 12 | 6 % |
| $S_5$ | CS | 29 | 15 % |
| $S_6$ | CS | 13 | 7 % |
| $S_7$ | TU | 66 | 35 % |
| $S_8$ | TU | 7 | 3,7 % |
| $S_9$ | TU | 10 | 5 % |
| $S_{10}$ | TU | 28 | 15 % |
| $S_{11}$ | TU | 10 | 5 % |
| $S_{12}$ | TU | 44 | 23 % |
| $S_{13}$ | TU | 39 | 21 % |
| $S_{14}$ | TU | 8 | 4 % |

Cuadro 7.6: Media, Mediana y Desviación Estándar de las técnicas

| | Media | Mediana | Desviación Estándar |
|---|---|---|---|
| CS | 15 | 11 | 13,9 |
| TU | 13,9 | 10 | 11,6 |

Las variaciones entre los datos obtenidos también pueden atribuirse a errores cometidos en la aplicación de las técnicas. Sin embargo, las revisiones semanales realizadas a las clases JUnit y a las notas entregadas no revelan anomalías en la aplicación de las técnicas. Por lo tanto, entendemos que todos los datos recolectados corresponden a una correcta aplicación de las técnicas y por lo tanto deben ser considerados, decidiendo entonces que ninguno debe ser eliminado de las observaciones.

Se observa en el Cuadro 7.6 que la técnica TU presenta medidas de tendencia central más bajas en comparación a CS, aunque la diferencia es poco significativa (1,1 y 1) en relación a los valores de la media y mediana. La desviación estándar de ambas técnicas es elevada, siendo de valores próximos a alcanzar la media y mediana, lo cual indica que los datos tienen una gran variabilidad encontrándose diseminados respecto al nivel central.

Al comparar estos resultados con los obtenidos en las experiencias iniciales para la efectividad, se visualizan algunas diferencias.

Ambas técnicas resultaron más efectivas en las experiencias iniciales, obteniendo valores de tendencia central superiores. En el caso de la media, CS presentó un valor que duplica al valor obtenido en este experimento, mientras que TU casi triplicó el valor correspondiente a este experimento.

El valor de la mediana para ambas técnicas fue triplicado en las experiencias iniciales. Además, la desviación estándar fue mayor para este experimento que para las experiencias iniciales.

El hecho de que las técnicas presentaran una efectividad mayor en las experiencias iniciales puede atribuirse a las diferencias entre los programas utilizados. El programa utilizado en las experiencias iniciales es un programa muy sencillo

y con pocas LOCs, mientras que el programa utilizado en este experimento es bastante más complejo y extenso.

Otra observación interesante es que la comparación de las medidas de tendencia entre ambas técnicas revela resultados opuestos entre ambos experimentos. En las experiencias iniciales TU fue la técnica más efectiva mientras que en este experimento lo es CS. Una causa de esta diferencia puede ser el número reducido de observaciones en ambos experimentos. Se necesitan más experimentos que dispongan de mayor cantidad de observaciones para obtener conclusiones al respecto.

A continuación se presentan las pruebas de hipótesis realizadas para comparar la efectividad de las técnicas.

Como se explica en el capítulo 2 para poder aplicar test paramétricos se requiere que las muestras se aproximen a una distribución normal, y para realizar pruebas de normalidad se debe contar con una cantidad mínima de observaciones. En el caso de nuestro experimento no se pueden realizar pruebas de normalidad ya que se tienen a lo sumo 8 observaciones para cada muestra, por lo cual no es posible aplicar pruebas paramétricas.

Se aplican entonces los test no paramétricos Kruskal-Wallis y Mann-Whitney. Se plantea la hipótesis nula que indica que las medianas de efectividad de ambas técnicas son iguales, junto con la hipótesis alternativa correspondiente:

$H_0 : \mu_{ECS} = \mu_{ETU}$
$H_1 : \mu_{ECS} <> \mu_{ETU}$

Se aplican ambos tests utilizando los datos de los defectos totales detectados por sujeto presentados en el cuadro 7.5. Se hacen los tests estadísticos sólo con la cantidad de defectos encontrados, y no con la efectividad calculada. De esta forma el resultado de los test es independiente de la cantidad de defectos totales estimada, eliminando el posible error asociado a la estimación. Esto es posible debido a que todas las observaciones corresponden a un mismo programa.

Para rechazar $H_0$ con una probabilidad de error aceptable se establece que esta no debe ser mayor al 10 %. Los resultados de la prueba Kruskal-Wallis se presentan en el cuadro 7.7, donde se muestra el estadístico obtenido en la prueba y la probabilidad de error resultante (valor de la tabla estadística).

Cuadro 7.7: Prueba de Kruskal-Wallis - Efectividad de CS y TU

| Chi-cuadrado | Probabilidad de Error |
|---|---|
| 0,341 | 75 % |

Como se observa en el cuadro, la probabilidad de error es mucho mayor al 10 %, por lo cual no es posible rechazar $H_0$. De lo anterior se deduce que **no existe evidencia estadística para afirmar que la efectividad de las técnicas CS y TU es diferente.**

Se aplica la prueba de Mann-Whitney para ver si se confirma el resultado anterior. Las hipótesis para Mann-Whitney son las mismas que para Kruskal-Wallis.

En el cuadro 7.8 se presentan los resultados de la prueba de Mann-Whitney. En el mismo se presenta el estadístico que se obtiene de la prueba (U), la cantidad de observaciones para cada muestra (n1 y n2), y la probabilidad de error

asociada.

Cuadro 7.8: Prueba de Mann-Whitney - Efectividad de CS y TU

| U Mann-Whitney | (n1 ; n2) | Probabilidad de Error |
|---|---|---|
| 19,5 | (8;6) | 33,1 % |

En base a los resultados obtenidos no es posible rechazar la hipótesis nula, debido a que la probabilidad de error es mayor al 10 %. **Con el test de Mann-Whitney también obtenemos que no existe evidencia estadística para afirmar que existe una diferencia de efectividad entre las técnicas CS y TU.**

En las experiencias iniciales se obtuvo el mismo resultado ya que ninguno de los test rechazó la hipótesis nula. De esto se deduce que las diferencias de efectividad que se aprecian no son significativas estadísticamente. Se necesitan más experimentos que posean mayor cantidad de observaciones para obtener resultados más significativos al respecto.

### Costo de las Técnicas

El costo de aplicar cada técnica se calcula como la sumatoria de los siguientes datos:

- **Tiempo de Diseño de los Casos de Prueba (TDCP)**: Tiempo que se invierte en el diseño y codificación en JUnit de los casos de prueba que cumplen con la técnica.
- **Tiempo de Ejecución de los Casos de Prueba (TE)**: Corresponde al tiempo de ejecución de los casos de prueba de la clase main, que es la encargada de implementar la interfaz con el usuario. Para los casos de prueba codificados en JUnit no se registra el tiempo de ejecución ya que se considera nulo.

Por lo tanto, el costo de las técnicas CS y TU se calcula de la siguiente forma:

$$Costo(CS) = TDCP_{CS} + TE_{CS}$$
$$Costo(TU) = TDCP_{TU} + TE_{TU}$$

En los Cuadros 7.9 y 7.10 se presenta el costo de aplicar las técnicas CS y TU por cada sujeto respectivamente. Las filas representan las clases que componen el programa, y las columnas representan los sujetos que aplican la técnica ($S_1$ a $S_6$ aplican CS y $S_7$ a $S_{14}$ aplican TU). El valor de una celda corresponde al tiempo en minutos que le llevó al sujeto diseñar y codificar los casos de prueba (y el tiempo de ejecutarlos si corresponde). La última fila del cuadro indica el tiempo total (costo) empleado por cada sujeto para todo el programa.

El Cuadro 7.11 presenta la media, mediana y la desviación estándar de cada técnica respecto al Costo.

Al igual que para la efectividad, aquí también se observan valores que distan significativamente de la media. En este caso, esto ocurre mucho más para TU que para CS. Consideramos que los motivos de la variabilidad son los mismos

Cuadro 7.9: Costo de aplicar CS en minutos

| Clase | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ |
|---|---|---|---|---|---|---|
| TiposDescuento | 55 | 20 | 52 | 35 | 13 | 30 |
| TiposCargo | 25 | 15 | 15 | 35 | 10 | 30 |
| Franjas | 90 | 35 | 62 | 45 | 15 | 40 |
| MaxMinCodigos | 20 | 15 | 17 | 15 | 10 | 30 |
| Descuento | 44 | 20 | 43 | 25 | 18 | 30 |
| ReciboSueldo | 98 | 80 | 81 | 60 | 50 | 30 |
| DescuentoRecibo | 36 | 15 | 28 | 20 | 10 | 30 |
| Funcionario | 94 | 75 | 76 | 50 | 44 | 60 |
| Cargo | 42 | 35 | 46 | 25 | 10 | 90 |
| Persistencia | 990 | 517 | 532 | 780 | 450 | 420 |
| Logica | 810 | 411 | 522 | 720 | 660 | 600 |
| Main | 940 | 528 | 737 | 480 | 580 | 120 |
| Ejecución Main | 35 | 89 | 270 | 180 | 280 | 600 |
| **Costo Total** | **3279** | **1855** | **2481** | **2470** | **2150** | **2110** |

Cuadro 7.10: Costo de aplicar TU en minutos

| Clase | $S_7$ | $S_8$ | $S_9$ | $S_{10}$ | $S_{11}$ | $S_{12}$ | $S_{13}$ | $S_{14}$ |
|---|---|---|---|---|---|---|---|---|
| TiposDescuento | 10 | 25 | 18 | 40 | 40 | 10 | 10 | 20 |
| TiposCargo | 10 | 25 | 18 | 22 | 27 | 10 | 9 | 20 |
| Franjas | 50 | 25 | 33 | 58 | 26 | 15 | 18 | 25 |
| MaxMinCodigos | 10 | 25 | 21 | 23 | 10 | 10 | 8 | 20 |
| Descuento | 30 | 30 | 30 | 59 | 26 | 15 | 14 | 25 |
| ReciboSueldo | 75 | 42 | 35 | 132 | 15 | 15 | 29 | 53 |
| DescuentoRecibo | 15 | 15 | 15 | 22 | 20 | 10 | 8 | 13 |
| Funcionario | 45 | 45 | 40 | 129 | 20 | 15 | 26 | 39 |
| Cargo | 30 | 25 | 20 | 64 | 25 | 10 | 12 | 27 |
| Persistencia | 780 | 640 | 360 | 685 | 412 | 450 | 672 | 795 |
| Logica | 1320 | 570 | 830 | 946 | 613 | 1380 | 1283 | 1710 |
| Main | 570 | 375 | 1080 | 1060 | 427 | 2110 | 1932 | 2650 |
| Ejecución Main | 270 | 40 | 210 | 40 | 110 | 75 | 2292 | 125 |
| **Costo Total** | **3215** | **1882** | **2710** | **3280** | **1771** | **4125** | **6313** | **5522** |

Cuadro 7.11: Media, Mediana y Desviación Estándar de las técnicas

| Técnica | Media | Mediana | Desviación Estándar |
|---|---|---|---|
| CS | 2390,8 | 2310 | 495,4 |
| TU | 3602,2 | 3247 | 1633,1 |

que para la efectividad, y como fue explicado anteriormente, no se van a eliminar observaciones ya que se considera que las técnicas fueron correctamente aplicadas por todos los sujetos.

Según las medidas de tendencia central, la técnica TU parecería ser más costosa que CS, lo que es esperado dadas las características de ambas técnicas, esta observación también es realizada en las experiencias iniciales. Los valores de media y mediana son superiores para TU, la media para esta técnica supera

a la de CS por 21,6 horas (1300 min) aproximadamente, y la mediana por unas 15 horas (900 min). Estas diferencias son significativas respecto a los valores de las medias y medianas. El valor de la desviación estándar para CS es de un quinto del valor de la media aproximadamente, mientras que la desviación estándar para TU es la mitad del valor de la media aproximadamente. Esto indica que los tiempos invertidos por los sujetos que aplicaron CS son mucho menos variables entre sí que los invertidos por los sujetos que aplicaron TU.

Se observa en los resultados obtenidos en las experiencias iniciales que también tuvo un mayor costo la aplicación de TU que la aplicación de CS. Sin embargo, la relación entre el costo de ambas técnicas difiere entre las experiencias iniciales y este experimento.

En en este experimento la media y la mediana para TU son 0,5 y 0,4 veces mayor que las correspondientes a CS respectivamente. En el caso de las experiencias iniciales los valores de media y mediana son 1,5 y 1,4 veces mayor para TU en comparación a CS respectivamente. Esto expone que en las experiencias iniciales la diferencia entre los costos de las técnicas es bastante mayor que la diferencia obtenida en este experimento. Es necesario realizar más experimentos con diferentes programas para obtener información acerca de si estas diferencias entre los costos de las técnicas están relacionadas a las características del programa bajo prueba (tamaño, complejidad, etc).

La diferencia entre las desviaciones estándar es un poco mayor para este experimento, siendo en ambos casos mayor la desviación de TU. En las experiencias iniciales la desviación estándar de TU es 1,9 veces mayor que la correspondiente a CS, mientras que para este experimento la diferencia es de 2,3 veces. Se deduce entonces que en ambos experimentos el costo de aplicar TU resulta ser mucho más variable entre los sujetos que el costo de aplicar CS.

Nuevamente se utilizan los test no paramétricos Mann-Whitney y Kruskal-Wallis. La hipótesis nula en este caso es que las medianas del costo de las técnicas son iguales:

$$H_0 : \mu_{CCS} = \mu_{CTU}$$
$$H_1 : \mu_{CCS} <> \mu_{CTU}$$

Los resultados de la prueba Kruskal-Wallis se presentan en el cuadro 7.12, donde se muestra el estadístico obtenido y la probabilidad de error resultante.

Cuadro 7.12: Prueba de Kruskal-Wallis - Costo de CS y TU

| Chi-cuadrado | Probabilidad de Error |
|---|---|
| 2,017 | 25 % |

El cuadro muestra que la probabilidad de error es mayor al 10 %, por lo cual no es posible rechazar $H_0$. Esto indica que **no existe evidencia estadística para afirmar que el costo de las técnicas es diferente**.

En el cuadro 7.13 se presentan los resultados de aplicar la prueba de Mann-Whitney. En el mismo se muestra el valor del estadístico obtenido, la cantidad de observaciones para cada muestra, y la probabilidad de error.

En este caso la probabilidad de error es menor al 10 %, por lo cual el test de Mann-Whitney rechaza la hipótesis nula. Esto significa que **según el test de**

Cuadro 7.13: Prueba de Mann-Whitney - Costo de CS y TU

| U Mann-Whitney | (n1 ; n2) | Probabilidad de Error |
|---|---|---|
| 13 | (8;6) | 9,1 % |

**Mann-Whitney existe evidencia estadistica para afirmar que el costo de ambas técnicas es diferente.**

La prueba de Kruskal-Wallis no asume normalidad en los datos dado que es una prueba no paramétrica. Sin embargo, sí asume que los datos vienen de la misma distribución. Una forma común en que se viola este supuesto es con datos heterocedásticos. Ante datos con un alto nivel de heterocedasticidad, el test de Kruskal-Wallis puede no rechazar la hipótesis nula cuando debería rechazarla (es decir, cuando existe una diferencia real entre las muestras).

Existe heterocedasticidad cuando no se cumple la homocedasticidad de los datos. Que los datos sean homocedásticos significa que las varianzas de las muestras son similares o iguales.

Para probar la homocedasticidad de los datos se suele utilizar el test de Levene. La hipótesis nula de este test es que las varianzas de las muestras son iguales. La hipótesis alternativa correspondiente indica que hay diferencias entre las varianzas de las muestras.

Se aplica el test de Levene para comprobar la homocedasticidad de los datos. El resultado obtenido rechaza la hipótesis nula con una probabilidad de error de 2,9 %. Esto indica que los datos son heterocedásticos con un 97,1 % de confianza, por lo cual puede suceder que Kruskal-Wallis no esté rechazando la hipótesis nula cuando debería hacerlo. Por esto no tendremos en cuenta que Kruskal-Wallis no rechace la hipótesis nula del costo de las técnicas.

**En base al resultado del test de Mann-Whitney se concluye que existe evidencia estadística que indica que TU es más costosa que CS.**

## 7.7. Comparación con Otros Experimentos

Es interesante poder realizar algún análisis comparativo entre nuestro experimento y los experimentos estudiados en el capítulo correspondiente al marco de comparación. Sin embargo, se encuentran algunas diferencias entre los experimentos estudiados y este experimento que dificultan el análisis.

En el experimento de Basili-Selby (B-S) se aplican las técnicas Partición de Clases de Equivalencia, Análisis de Valores Límite, Cubrimiento de Sentencias y Revisión de Código. En el experimento de Kamsties-Lott (K-L) las técnicas utilizadas son Partición de Clases de Equivalencia, Análisis de Valores Límite, Criterio de Cubrimiento de Ramificaciones, Ciclos, Operadores Relacionales, y Revisión de Código. Los investigadores Macdonal-Miller (M-M) utilizan Inspecciones basadas en papel y basadas en la herramienta ASSIST, en su experimento. En los experimentos de Juristo-Vega (J-V) se utiliza Partición en Clases de Equivalencia, Revisión de Código y un criterio lo más cercano a Cubrimiento de Decisión. Observamos que ninguno de los experimentos estudiados aplica TU, por lo cual no podemos realizar comparaciones con respecto a esta técnica. Sólo el experimento de B-S aplica la técnica CS.

Entre los lenguajes de los programas utilizados se encuentran: Fortran, Simple-T, C y C++. Ningún programa utilizado por los investigadores está codificado

en Java.

En todos los casos, los programas utilizados en los experimentos son mucho más pequeños en tamaño que el utilizado en este experimento. El programa más grande es uno de los utilizados por B-S que contiene 365 LOCs. El programa Contabilidad utilizado en este experimento tiene un tamaño de 1820 LOCs aproximadamente.

Otra característica de los programas utilizados por los distintos investigadores es que los mismos contienen faltas inyectadas. En cambio, el programa utilizado en este experimento sólo contiene las faltas cometidas al construir el mismo.

En cuanto a la cantidad de sujetos que participan en los experimentos, en todos los casos es una muestra bastante mayor en comparación a nuestro experimento. En el experimento de B-S participan 74 sujetos, en K-L 50, en M-M 43, y en J-V 196 y 46 en su experimento I y II respectivamente. El experimento presentado en este capítulo es ejecutado solamente por 14 sujetos.

Notamos de esta forma que son varios los aspectos que diferencian a nuestro experimento de los estudiados.

No podemos comparar el costo de aplicar CS en B-S y nuestro experimento, debido a que el costo varía dependiendo del programa. Es necesario comparar con un experimento que utilice el mismo programa que nosotros, lo que nos impide realizar la comparación con B-S.

En lo que respecta a la efectividad, B-S plantea la media expresada en cantidad de defectos encontrados y no como porcentaje. No podemos comparar la cantidad de defectos encontrados dado que estos dependen del programa. Mientras los programas de Basili contienen entre 6 y 9 defectos, el nuestro tiene 1820 LOCs. Se necesita poder comparar la efectividad en porcentaje, sin embargo, Basili no presenta en su artículo este dato, ni tampoco presenta todas las observaciones por lo que no lo podemos calcular. Debido a esto no es posible realizar la comparación en este sentido.

Podemos comparar algunas decisiones tomadas en los experimentos. En todos los experimentos se tuvo en cuenta la amenaza de aprendizaje obtenido, es decir, en ningún caso el sujeto aplica la misma técnica, ni verifica el mismo programa más de una vez. Dicha decisión también es tomada en el diseño de este experimento. En la mayoría de los experimentos el nivel de experiencia de los sujetos podría considerarse similar al nivel de nuestro experimento. En el único experimento en el cual se consideran diferentes niveles de experiencia de los sujetos es en el realizado por B-S, donde participan sujetos avanzados, intermedios y junior. En la mayoría de los experimentos se decide clasificar las faltas detectadas, sólo el experimento de M-M no las clasifica. En el caso de este experimento se decidió no realizar la clasificación de las faltas. El motivo de esta decisión es que consideramos que es bastante costosa la clasificación de cada defecto. Esto se retoma en el capítulo 8, en trabajos a futuro.

## 7.8. Encuestas a los Sujetos

Una vez finalizado el experimento se envía un correo electrónico a los sujetos informando la finalización satisfactoria del trabajo. Además, se comenta los aportes que el trabajo le brinda al proyecto de grado de las tutoras, y por consiguiente, la importancia de su participación en el mismo. Por último se realizan

algunas preguntas a los mismos para conocer su opinión sobre la materia cursada en lo referente al aporte a nivel personal, las clases brindadas, la metodología seguida a lo largo del curso, y la carga horaria.

Las preguntas realizadas junto con algunas respuestas representativas se presentan a continuación:

1. ¿Consideran que fue provechoso para ustedes participar del MT [1]?

   - Sí considero que fue provechoso. En particular me quedo con la idea de cómo funcionan las técnicas y para un futuro por ahí me sirve poder aplicarlas.
   - La verdad que sí, laboralmente me abrió un poco el panorama del testing, y a nivel de facultad fue una materia que si bien se tuvo que trabajar duro en algún momento, no se me hizo pesada.
   - Si, estuvo bueno, en particular nunca había aplicado formalmente ninguna técnica de verificación, me sirvió, por un lado reafirmé los conocimientos acerca del tema que venían de Introducción a la Ingeniería de Software, y por otro conocí y utilicé la herramienta JUnit por primera vez, me resultó una herramienta muy potente para verificación en java.

2. ¿Qué opinión tienen de las clases brindadas en facultad?

   - Me pareció que las clases estuvieron super bien dictadas, nos dió para comprender bastante bien cómo es que se debería aplicar cada una de las técnicas dadas. A su vez, los ejercicios prácticos que se realizaron durante las clases ayudaron a comprenderlas mejor.
   - Cansadoras de tan largas, pero mejor, así no teníamos que ir otros días, se entendió bien claro todo y con los ejercicios que hicimos no se precisa más.
   - Las clases me parecieron correctas ya que vimos tanto el teórico, como lo necesario de práctico para poder realizar las técnicas sin mayor necesidad de preguntar para evacuar las dudas.

3. ¿Qué opinión tienen del desarrollo del MT durante estas semanas?

   - Me parece que estuvo todo bastante bien. Silvana y Carmen respondieron a todas las preguntas rapidísimo.
   - Seguimos un procedimiento claramente pautado al inicio y se siguió perfecto durante todo el MT, cuando solicité correcciones o tenía dudas siempre me encontré con las respuestas inmediatas por parte de ustedes.
   - Impecable. Fue muy buena la disposición para responder a los diferentes inconvenientes que se fueron planteando.

4. ¿Consideran que aplicarían alguna de las técnicas en su ámbito laboral?

   - Sin duda lo sugeriría para que alguien más lo haga.
   - Realmente creo que Inter e Intra no los utilizaría pero sí Sentencias o alguna otra técnica más rápida. También creo que podría ser posible usar Inter-Intra pero con análisis de código o alguna otra forma de acelerar el proceso. Además en funcionalidades muy complejas resulta un poco complicado utilizar intra. Otra alternativa es construir una herramienta que arme el grafo (no parece muy complicado) y armar los casos a partir de

[1] Módulo de Taller (MT) es la materia cursada por los sujetos en el marco del experimento.

los grafos generados.
- Es difícil, tal vez para algún procedimiento en particular que estuviera dando muchos problemas. El tema es que los tiempos estimados en los proyectos no consideran muy bien el tiempo de desarrollo de pruebas unitarias. También depende del sistema, para sistemas complejos puede llegar a ser imposible.

5. ¿Consideran que la carga de trabajo fue acorde?

   - En promedio me parece que sí, quizá algunos picos pero en promedio bien.
   - La carga de trabajo me pareció justa.
   - Aunque en las últimas semanas, al menos en mi caso, me tomó más tiempo de lo esperado, eso compensaría el tiempo que me tomó en las primeras semanas que fue mucho menor, por lo que sí me parece que en total la carga de trabajo fue la esperada.

En el cuadro 7.14 se observan las respuestas de los 14 sujetos para las preguntas 1, 4 y 5 anteriormente planteadas. Si bien las respuestas reales son más extensas, son repuestas bastantes cerradas por lo cual es posible asociarlas a un "sí" o a un "no". Las otras dos preguntas (2 y 3) son muy abiertas, por lo que no fue posible presentarlas de la misma forma. Las respuestas de todos los sujetos se presentan en el Anexo C.

Cuadro 7.14: Respuestas de los sujetos

| Sujeto | Pregunta 1 | Pregunta 4 | Pregunta 5 |
|---|---|---|---|
| $S_1$ | Sí | No | No |
| $S_2$ | Sí | Sí | Sí |
| $S_3$ | Sí | Sí | No |
| $S_4$ | Sí | No | Sí |
| $S_5$ | Sí | Sí | Sí |
| $S_6$ | Sí | No | No |
| $S_7$ | Sí | No | Sí |
| $S_8$ | Sí | No | Sí |
| $S_9$ | Sí | Sí | Sí |
| $S_{10}$ | Sí | Sí | No |
| $S_{11}$ | Sí | No | Sí |
| $S_{12}$ | Sí | No | Sí |
| $S_{13}$ | Sí | Sí | Sí |
| $S_{14}$ | Sí | Sí | Sí |

Las respuestas obtenidas por parte de los sujetos fueron satisfactorias. En líneas generales, indicaron que la materia les resultó provechosa a nivel personal. Coincidieron en que las clases brindadas fueron adecuadas, aunque algunos opinaron que la carga horaria de las mismas fue muy extensa. Con respecto a la metodología seguida durante el curso se mostraron conformes. Consideramos que la información obtenida mediante esta encuesta es de valor para identificar aspectos positivos y oportunidades de mejora para futuros experimentos.

## 7.9. Conclusiones

En este capítulo se presenta un experimento formal con el objetivo de comparar la efectividad y el costo de las técnicas de verificación: Cubrimiento de Sentencias (CS) y Todos los Usos (TU).

Se construye una planilla de registro de defectos y tiempos, y se realizan mejoras a la Guía de verificación a ser seguida por los sujetos. Estos materiales de soporte son de gran aporte para futuros experimentos que realice el Grupo de Ingeniería de Software y otros grupos de investigación.

Se prepara y brinda una clase teórico - práctica que capacita a los sujetos en la aplicación de las técnicas y en el uso de los materiales de soporte. Se complementa la clase con un entrenamiento que consiste en la aplicación de las técnicas sobre un programa simple.

La planificación realizada para ejecutar el experimento en 8 semanas resultó exitosa. Se logró que todos los sujetos participantes culminaran las pruebas del programa en el tiempo establecido. Consideramos que la exigencia de una entrega obligatoria por semana y el seguimiento constante a los sujetos contribuyó al cumplimiento de la planificación.

Cuando se realizan experimentos es complejo lograr cumplir con los tiempos planificados para su ejecución, dado que pueden afrontarse diversas complicaciones. El Experimento Año 2008 no logró ser ejecutado en el tiempo estimado. Se planificó que tuviera una duración de un semestre, pero diferentes inconvenientes provocaron que su duración se extendiera a un año.

Se realizó una revisión de las entregas efectuadas, permitiendo controlar el correcto llenado de la planilla Grillo y una adecuada aplicación de las técnicas.

Se construye una planilla que contiene los defectos totales detectados del programa, discriminados por clase.

También se construye una planilla con los defectos encontrados por cada sujeto. Dicha planilla se encuentra discriminada por clase, y registra el tiempo en minutos que le llevó al sujeto la detección de cada defecto. Esta información no fue analizada en el experimento, sin embargo, puede ser útil para investigar qué tipos de defectos resultan más o menos costosos de detectar.

Los datos obtenidos, tanto para la efectividad como para el costo, presentan variabilidad respecto a las medidas de tendencia central. Consideramos que un posible motivo de las diferencias entre los datos es la variabilidad entre las características de los sujetos, por ejemplo, sus habilidades, motivación, y dedicación. Además, observamos que algunos sujetos fueron más detallistas al realizar la verificación que otros, estando esto relacionado a la personalidad de los sujetos y no a las técnicas en sí mismas.

Los resultados obtenidos con los test de hipótesis indican que no existe evidencia estadística para afirmar que una técnica es más efectiva que la otra. En las experiencias iniciales se obtuvo el mismo resultado ya que ninguno de los test rechazó la hipótesis nula. De esto se deduce que las diferencias de efectividad que se aprecian no son significativas estadísticamente.

Respecto a la comparación del costo de las técnicas, se observa que el test de Kruskal-Wallis no rechaza la hipótesis nula. En cambio, el test de Mann-Whitney rechaza dicha hipótesis.

Se aplica el test de Levene para probar la homocedasticidad de los datos. El resultado obtenido rechaza la hipótesis nula con una probabilidad de error de 2,9 %. Esto indica que puede suceder que Kruskal-Wallis no esté rechazando

la hipótesis nula cuando debería hacerlo. Por esto no se tiene en cuenta que Kruskal-Wallis no rechace la hipótesis nula del costo de las técnicas.

En base al resultado del test de Mann-Whitney se concluye que existe evidencia estadística que indica que TU es más costosa que CS.

En las experiencias iniciales ambos test rechazaron la hipótesis nula, indicando que existe evidencia estadística para concluir que TU es más costosa que CS.

# Capítulo 8

# Conclusiones y Trabajos a Futuro

Los principales logros de este proyecto son:

- La construcción de un marco de comparación de experimentos formales que evalúan técnicas de verificación, y su aplicación en cuatro experimentos formales conocidos.
- El estudio realizado sobre la aplicación de las técnicas de Cubrimiento de Sentencias (CS) y Todos los Usos (TU). En base a dicho estudio se definieron las técnicas a utilizar en el experimento.
- La preparación y dictado de la capacitación a los sujetos que participan del experimento. En esta actividad se construyen materiales de soporte que pueden ser útiles también para futuros experimentos.
- La definición, ejecución y análisis de datos de un experimento formal que permite comparar la efectividad y el costo de las técnicas CS y TU.

## 8.1. Conclusiones

Se realiza un estudio sobre Ingeniería de Software Empírica, abarcando conceptos básicos de experimentación en el área, fases que componen a los experimentos (objetivos, diseño, operación y análisis de datos), y terminología (unidad experimental, parámetros, factores, alternativas, variables de respuesta, entre otros). Este estudio resultó indispensable para adquirir conocimientos acerca de los fundamentos de Ingeniería de Software Empírica que nos permitieran cumplir con los objetivos.

El estudio realizado sobre experimentos formales en Ingeniería de Software que evalúan técnicas de verificación, en el cual se analizaron cuatro experimentos, revela que en la mayoría de los casos la información expuesta no es suficiente. Se detecta que los programas utilizados no son presentados por los autores y no se definen correctamente las técnicas aplicadas. Consideramos que esta falta de información dificulta replicar y comparar los experimentos.

Otra observación sobre los experimentos estudiados es que en la mayoría de los casos las faltas presentes en los programas son inyectadas en el código. Creemos que esta práctica le resta realismo a las condiciones de ejecución del experimento en comparación con la práctica real de la industria.

Se elabora un marco de comparación de experimentos formales que evalúan técnicas de verificación, con el cual se busca proveer formalidad al momento de comparar distintos experimentos. Al aplicar el marco se obtiene una mayor comprensión de los principales aspectos de los experimentos y se logra compararlos a través de sus características relevantes. Este marco es utilizado para comparar los cuatro experimentos formales que se estudiaron.

Para alcanzar el objetivo de comparar la efectividad y el costo de las técnicas CS y TU se realiza un experimento formal.

Al realizar el diseño del experimento se tuvo especial cuidado en ciertas características que pueden influir en los resultados. Se seleccionan sujetos con similares características, siendo todos estudiantes avanzados de la carrera de Ingeniería en Computación de la Facultad de Ingeniería. Además, se homogeniza el grupo mediante clases niveladoras (capacitación). Los sujetos desconocen que forman parte de un experimento, participando del mismo en el marco de una materia de Facultad con una cierta cantidad de créditos asignados. Por ello su motivación está asociada solamente a la aprobación del curso. De esta forma se evitan desvíos que pueden producirse en el trabajo de los sujetos por sentirse observados.

Las técnicas utilizadas en el experimento son CS y TU. Ambas técnicas son dinámicas de caja blanca, siendo CS basada en el flujo de control del programa y TU basada en el flujo de datos del mismo. La técnica TU es muy poco conocida y aplicada, por lo que requirió de un análisis profundo. Se consultó la literatura para obtener una definición apropiada de ambas técnicas.

Cada sujeto aplica sólo una de las técnicas, evitando así el efecto de aprendizaje. Como la cantidad de sujetos que se dispone para el experimento es pequeña se decide aplicar las técnicas sobre un único programa.

Los sujetos son capacitados en la aplicación de las técnicas y en el uso de los materiales de soporte. Para ello se prepararon y dictaron clases teórico-prácticas, y se realizó un entrenamiento en el cual los sujetos practicaron la aplicación de las técnicas sobre un programa sencillo. Los materiales creados para esta actividad pueden ser útiles también para futuros experimentos.

La capacitación es realizada en dos instancias (llamadas experiencias iniciales) ya que los sujetos fueron divididos en dos grupos. La primer experiencia es muy útil para adquirir información valiosa sobre la adecuación de la capacitación brindada a los estudiantes, y mejorarla para la segunda experiencia. Además, otorga una visión general de las dificultades afrontadas en la aplicación de las técnicas.

Si bien el objetivo de estas experiencias es brindar la capacitación a los sujetos, las mismas son consideradas un experimento en sí mismo. Este experimento consiste en 21 sujetos aplicando las técnicas CS y TU para verificar un programa sencillo escrito en Java. A 10 sujetos se asigna la aplicación de CS y a 11 la aplicación de TU.

El programa utilizado durante las experiencias iniciales es muy pequeño y sus defectos son inyectados en el código. De todas formas, consideramos que para la capacitación fue adecuado, ya que logró el cometido de que los sujetos aprendieran a registrar los defectos en la planilla Grillo, seguir el proceso de

verificación establecido en la Guía, y aplicar las técnicas de verificación.

Los datos recolectados son analizados aplicando las pruebas de hipótesis de Kruskal-Wallis y Mann-Whitney. Los resultados de dicho experimento indican que no existe evidencia estadística para afirmar que una técnica es más efectiva que la otra. Respecto al costo, los test revelan que existe evidencia estadística que indica que la técnica TU es mas costosa que CS, como es esperado dadas las características de ambas técnicas.

Una vez finalizada la capacitación de los sujetos se lleva a cabo el experimento formal que constituye el foco central de este proyecto.

En el experimento participa un subconjunto de los sujetos pertenecientes a las experiencias iniciales, de los cuales 6 aplican CS y 8 aplican TU. La ejecución del experimento se realiza en una única instancia con una duración de 8 semanas.

Este experimento es de mayor porte que las experiencias iniciales con respecto al programa verificado, siendo el programa utilizado en este experimento más realista respecto a la práctica de la industria. Normalmente los programas usados en los experimentos que se han realizado en el área resultan ser muy pequeños y sencillos, lo cual no representa el tipo de software que se genera en la industria. Además, los defectos no son los cometidos por los desarrolladores, sino que son inyectados por los investigadores. Para el experimento formal se utiliza un programa construido específicamente para experimentos de este tipo. Fue construido para el Experimento Año 2008, y entre sus principales características se encuentra que sus faltas no son inyectadas en el código sino que son las cometidas por su desarrollador.

Los datos obtenidos en este experimento, tanto para la efectividad como para el costo, presentan variabilidad respecto a las medidas de tendencia central. Consideramos que un posible motivo de las diferencias entre los datos es la variabilidad entre las características de los sujetos, por ejemplo, sus habilidades, motivación, y dedicación. Además, observamos que algunos sujetos fueron más detallistas al realizar la verificación que otros, estando esto relacionado a la personalidad de los sujetos y no a las técnicas en sí mismas.

Los resultados obtenidos con los test indican que no existe evidencia estadística para afirmar que una técnica es más efectiva que la otra. En las experiencias iniciales se obtuvo el mismo resultado ya que ninguno de los test rechazó la hipótesis nula. De esto se deduce que las diferencias de efectividad que se aprecian no son significativas estadísticamente. Se necesitan más experimentos que posean mayor cantidad de observaciones para obtener resultados más significativos.

Respecto a la comparación del costo de las técnicas, se observa que el test de Kruskal-Wallis no rechaza la hipótesis nula, lo que significa que no se puede asegurar nada respecto a la relación entre el costo de las técnicas. En cambio, el test de Mann-Whitney rechaza dicha hipótesis, por lo tanto según este test existe evidencia estadística que indica que TU es más costosa que CS. Observamos que los resultados de ambos test son contradictorios.

Se aplica el test de Levene para probar la homocedasticidad de los datos. El resultado obtenido es que se rechaza la hipótesis nula con una probabilidad de error de 2,9 %. Esto indica que los datos son heterocedásticos con un 97,1 % de confianza, por lo cual puede suceder que Kruskal-Wallis no esté rechazando la hipótesis nula cuando debería hacerlo. Por esto no tendremos en cuenta que Kruskal-Wallis no rechaza la hipótesis nula del costo de las técnicas.

En base al resultado del test de Mann-Whitney se concluye que existe evidencia estadística que indica que TU es más costosa que CS. También en las

experiencias iniciales se concluye que TU es más costosa que CS.

Entendemos que es necesario realizar más experimentos, principalmente con más observaciones. También es interesante variar la complejidad de los programas probados. Si bien en el segundo experimento se utilizó un programa mucho más complejo y extenso que en el primero, no se logró aumentar la cantidad de observaciones, incluso dicha cantidad se redujo. Es necesario realizar experimentos con mayor número de observaciones.

Una vez finalizado el experimento se realizó una encuesta a los sujetos para averiguar su opinión sobre la materia cursada en lo referente a su aporte a nivel personal, las clases brindadas, la metodología seguida a lo largo del curso, y la carga horaria. Las respuestas obtenidas por parte de los sujetos fueron satisfactorias. En líneas generales, indicaron que la materia les resultó provechosa a nivel personal. Coincidieron en que las clases brindadas fueron adecuadas, aunque algunos opinaron que la carga horaria de las mismas fue muy extensa. Se mostraron conformes con la metodología seguida durante el curso. Consideramos que la información obtenida mediante esta encuesta es de valor para identificar aspectos positivos y oportunidades de mejora para futuros experimentos.

Algunos resultados obtenidos en el proyecto constituyen aportes al Grupo de Ingeniería de Software. En particular identificamos: el estudio realizado sobre la técnica TU, el marco de comparación de experimentos, el material preparado para el dictado de la capacitación (Módulos de Taller) y principalmente la experiencia en la realización de un experimento formal.

Es importante mencionar la información adquirida en la realización del experimento respecto a los recursos invertidos, de forma de brindar datos que permitan realizar estimaciones para próximos experimentos.

La capacitación de los sujetos requirió de 3 semanas para cada grupo. La primer semana se dedicó al estudio individual de JUnit y la realización de una tarea de aplicación sencilla. El sábado siguiente se dictó la clase teórico-práctica en el correr del día, teniendo una duración de 10 horas. La semana posterior se destinó al repaso individual de la clase y al envío de dudas. La jornada de aplicación de las técnicas a un simple programa escrito en Java fue llevada a cabo el siguiente sábado, teniendo también una duración de 10 horas.

Consideramos que los tiempos dedicados en la capacitación fueron adecuados, aunque entendemos que las jornadas de 10 horas fueron extensas y cansadoras. Es recomendable considerar distribuir la carga horaria entre más clases. En la primer instancia se capacitó a un grupo de 10 estudiantes y en la segunda a un grupo de 11. La cantidad de sujetos nos resultó manejable, no identificamos dificultades en este sentido.

La ejecución del experimento se desarrolló durante 8 semanas, durante las cuales los sujetos aplicaron una de las técnicas estudiadas a un programa escrito en Java. Este programa es bastante más complejo que el utilizado en la capacitación, estando constituido por 1820 LOCs distribuidos en 14 clases. Además, cuenta con una pequeña base de datos gestionada con el manejador HSQLDB. Los tiempos invertidos por los sujetos en esta instancia se ubican en el rango entre 30,9-54,6 horas en el caso de CS y entre 29,5-105,2 horas para TU. Estos tiempos solo consideran el diseño, codificación y ejecución de los casos de prueba, debe tenerse en cuenta también el tiempo invertido en comparar los resultados obtenidos con los esperados, tiempo de detección de los defectos en el código, realizar los pedidos de correcciones, etc.

En esta instancia participaron 14 de los 21 sujetos capacitados. Considera-

mos que la atención de 14 sujetos por parte de 2 personas, si bien es posible de realizar, implica una inversión de tiempo de alrededor de 3 horas diarias por parte de cada investigador. Deben atenderse numerosos pedidos de correcciones de los defectos detectados y dudas en general. Estos pedidos llegan durante el transcurso de la semana pero se intensifican durante los fines de semana. Además, debe realizarse el envío de las clases a verificar cada semana y controlar las entregas efectuadas. Estos controles implican corroborar que se entregue todo lo solicitado, y se complete la planilla de defectos de forma adecuada, principalmente describiendo de forma satisfactoria los defectos encontrados. También se llevaron a cabo revisiones de entregas seleccionadas al azar en cada semana, para verificar que se aplicaran las técnicas de forma adecuada.

Consideramos que la inversión de 8 semanas para la realización de la ejecución del experimento fue acertada, ya que se pudo finalizar en dicho tiempo y los estudiantes invirtieron una carga horaria similar a la esperada, en general.

Destacamos también la experiencia personal obtenida a lo largo del proyecto de grado. Logramos aprender a buscar, seleccionar y leer un gran numero de artículos para conocer el tema estudiado y lograr definir las técnicas de verificación en forma completa. Con esta experiencia adquirida llevamos a cabo la redacción del artículo *Towards a framework to compare formal experiments that evaluate verification techniques*, para la Conferencia Internacional en México de Ciencias de la Computación.

Parte de la experiencia personal es también la realización de los experimentos. Estudiar diversos experimentos realizados en la academia y en la industria nos permitieron conocer y aprender sobre las lecciones aprendidas de varios investigadores. Además, consideramos muy provechoso a nivel personal la preparación y dictado de los Módulos de Taller. Desempeñar un rol docente en un curso resultó ser muy interesante y enriquecedor.

A lo largo del experimento se fueron construyendo diversas planillas de registro de datos. La primer planilla consiste en la descripción, línea de código y clase Java de cada defecto detectado en el programa Contabilidad por los sujetos. Otra planilla disponible consta de el tiempo que invirtió cada sujeto en detectar la falta luego de observada la falla. Consideramos de gran valor los datos incluidos en estas planillas para poder llevar a cabo trabajos a futuro, y discutir sobre la efectividad y el costo según el tipo de defecto.

Se cuenta además con los casos de prueba JUnit construidos por los estudiantes durante cada semana de entrega. Se dispone por cada semana de los casos de prueba diseñados para verificar la parte del programa correspondiente a dicha semana. Cada sujeto entregó también los grafos y notas realizadas en papel que fueron necesarias para la construcción de los casos aplicando la técnica asignada. Consideramos que esta información puede ser analizada para conocer entre otras cosas, si la cantidad y complejidad de los casos de prueba generados varían según la técnica utilizada. También es posible analizar si los casos diseñados para CS cumplen también TU y viceversa. Las notas realizadas en papel permiten además detectar errores en la aplicación de las técnicas (grafos, identificación de definiciones y usos, generación de caminos, etc.) y conocer que tipos de errores son más comunes de cometer con TU y cuales con CS.

## 8.2. Trabajos a Futuro

Identificamos algunos temas interesantes como trabajos a futuro.

**Analizar la efectividad por tipo de defecto:**

Se cuenta con la información contenida en la planilla de defectos totales detectados del programa Contabilidad. Esta planilla contiene la unión de los defectos encontrados durante este experimento y el llevado a cabo en el Experimento Año 2008. Los defectos se encuentran separados por clase Java y con información suficiente para identificarlo en el código. Dichos defectos podrían clasificarse según alguna taxonomía que contemple los diferentes tipos de defectos. Con esta información es posible analizar la efectividad de las técnicas según los tipos de defectos.

**Analizar el costo de detección por tipo de defecto:**

Entre los datos recolectados en el experimento se encuentra el tiempo invertido por cada sujeto en detectar cada defecto. Este tiempo corresponde al tiempo que tarda el sujeto en detectar la falta luego de observada la falla. Esta información no fue analizada en este experimento, pero resulta interesante utilizar dichos datos para conocer si el costo de detección de defectos depende del tipo de defecto.

**Analizar los casos de prueba diseñados por los sujetos:**

En este trabajo no se analizó la información de los casos de prueba JUnit entregados por los sujetos. Se dispone de las clases JUnit entregadas por cada sujeto correspondientes a los casos de prueba diseñados para verificar el programa. También se cuenta con las anotaciones realizadas para aplicar las técnicas (grafos de flujo de control, caminos identificados, etc). Este material puede ser utilizado para comparar diferentes características de los casos de prueba generados en la aplicación de ambas técnicas, por ejemplo, puede resultar interesante analizar si la cantidad y complejidad de los casos de prueba generados varian según la técnica utilizada.

# Bibliografía


[1] Cecilia Apa. *Diseño y Ejecución de un Experimento con 5 Técnicas de Verificación Unitaria, Proyecto de Grado de Ingeniería en Computación, Facultad de Ingeniería, Universidad de la República*. 2009. 1.2

[2] Victor R. Basili and Richard W. Selby. Comparing the effectiveness of software testing strategies. *IEEE Transactions on software engineering*, 13(12):1278–1296, 1987. 1.2, 3

[3] Stephanie de León and Rosana Robaina. *Análisis de la efectividad y el costo de 5 técnicas de verificación, Proyecto de Grado de Ingeniería en Computación, Facultad de Ingeniería, Universidad de la República*. 2009. 1.2

[4] Norman E. Fenton and Shari L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach, Revised*. Course Technology, February 1998. 2.1

[5] Phyllis G. Frankl and Elaine J. Weyuker. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, 14(10):1483–1498, 1988. 4.2, 4.2, 4.2.1, 4.2.2

[6] Fernanda Grazioli. *Análisis de taxonomías de defectos, Proyecto de Grado de Ingeniería en Computación, Facultad de Ingeniería, Universidad de la República*. 2009. 1.2

[7] Mary Jean Harrold and Gregg Rothermel. Performing data flow testing on classes. *SIGSOFT Softw. Eng. Notes*, 19(5):154–163, 1994. 4.2, 4.2.2, 4.2.2

[8] Mary Jean Harrold and Mary Lou Soffa. Interprocedual data flow testing. In *TAV3: Proceedings of the ACM SIGSOFT '89 third Symposium on Software Testing, Analysis, and Verification*, pages 158–167, New York, NY, USA, 1989. ACM. 4.2, 4.2.2

[9] Natalia Juristo and Ana M. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001. 1.4, 2.1

[10] Natalia Juristo and Sira Vegas. Functional testing, structural testing, and code reading: What fault type do they each detect? *Empirical Methods and Studies in Software Engineering*, 2765/2003:208–232, 2003. 1.2, 3

[11] Erik Kamsties and Christopher M. Lott. An empirical evaluation of three defect-detection techniques. In *Proceedings of the Fifth European Software Engineering Conference*, pages 362–383, 1995. 1.2, 3

[12] F. Macdonald and J. Miller. A comparison of tool-based and paper-based software inspection. *Empirical Software Engineering*, 3(3):233–253, 1998. 1.2, 3

[13] E. J. Martínez. Notas del curso de posgrado *Maestría en Estadística Matemática*. In *Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires*, 2004. 2.4.5

[14] Ana Moreno, Forrest Shull, Natalia Juristo, and Sira Vegas. A look at 25 years of data. *IEEE Software*, 26(1):15–17, Jan.–Feb. 2009. 1.1

[15] Glenford J. Myers. A controlled experiment in program testing and code walkthroughs/inspections. *Communications of the ACM*, 21(9):760–768, September 1978. 1.1

[16] Peter Nelson, Marie Coffin, and Karen Copeland. *Introductory statistics for engineering experimentation*. Elsevier Science, California, 2003. 2.4.5

[17] M.R. Spiegel. *Estadística - 2da Edición*. Mc.Graw-Hill, Madrid, 1991. 2.4.5

[18] Jorge Triñanes. Construcción de un banco de pruebas de modelos de proceso. In *Proceedings de la IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento*, 2004. 2.1

[19] Diego Vallespir, Cecilia Apa, Stephanie De León, Rosana Robaina, and Juliana Herbert. Effectiveness of five verification techniques. In *Proceedings of the XXVIII International Conference of the Chilean Computer Society*, 2009. 1.2, 5

[20] Diego Vallespir, Fernanda Grazioli, and Juliana Herbert. A framework to evaluate defect taxonomies. In *Proceedings del XV Congreso Argentino de Ciencias de la Computación*, 2009. 1.2

[21] Diego Vallespir and Juliana Herbert. Effectiveness and cost of verification techniques: Preliminary conclusions on five techniques. In IEEE-Computer-Society, editor, *Proceedings of the Mexican International Conference in Computer Science*, 2009. 1.2, 5, 6.7.1

[22] Diego Vallespir, Silvana Moreno, Carmen Bogado, and Juliana Herbert. Towards a framework to compare formal experiments that evaluate verification techniques. In *Proceedings of the X Mexican International Conference in Computer Science*, 2009. 1.2

[23] Carolina Valverde and Bruno Bianchi. *Un caso de estudio en Calidad de Datos para Ingeniería de Software Empírica, Proyecto de Grado de Ingeniería en Computación, Facultad de Ingeniería, Universidad de la República*. 2009. 1.2

[24] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. 2.1, 2.4.5

# Apéndice A

# Clase Teórica

En este anexo se presenta la clase teórica brindada como capacitación a los sujetos. La clase contiene:

- Un breve repaso de algunos conceptos de Ingenería de Software.
- La explicación de la técnica Cubrimiento de Sentencias con ejemplos.
- Una introducción a *Data Flow Testing* y los criterios que se deben tener en cuenta.
- La explicación de la técnica Todos los Usos con ejemplos.
- Una introducción a la planilla de registro de defectos Grillo.

# Técnicas de Verificación

## Errores, Faltas y Fallas

un error humano → puede generar → una falta (interna) → que puede generar → una falla (externa) ?!

Nuestro objetivo: Descubrir defectos (faltas)

## ¿ Qué es Verificación y Validación ?

- **Verificación:** ¿Estamos construyendo el producto correctamente?
  Se comprueba que el software cumple los requisitos funcionales y no funcionales de su especificación.
- **Validación:** ¿Estamos construyendo el producto correcto?
  Comprueba que el software cumple las expectativas que el cliente espera.

## Técnicas de caja blanca

- Se llevan a cabo una vez que se escribe código.
- En el caso de las técnicas de caja blanca dinámicas buscan faltas ejecutando el código.
- Permite evaluar los valores de las variables, las constantes y los tipos de datos y si éstos son usados en el contexto en que se definieron.

## Técnicas de caja blanca (2)

- Basadas en el **flujo de control** del programa

  Expresan los cubrimientos del testing en términos del grafo de flujo de control del programa
- Basadas en el **flujo de datos** del programa

  Expresan los cubrimientos del testing en términos de las asociaciones definición-uso del programa

## Criterio de cubrimiento de sentencias

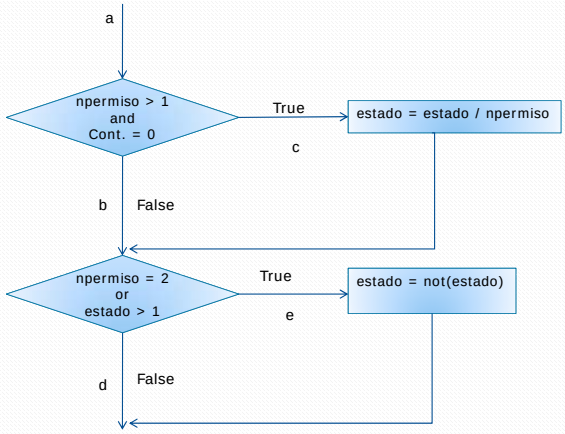
- Asegura que cada instrucción del código se ejecuta al menos una vez en el conjunto de casos de pruebas (CCP) construido.

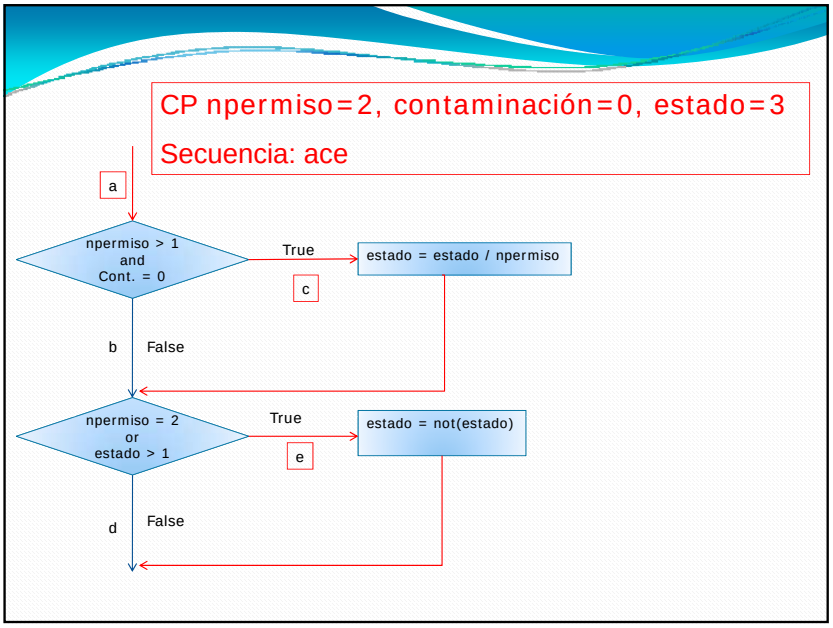
# Ejemplo 1

```
If (npermiso > 1) and (contaminación = 0)
{
    estado = estado / npermiso
}
If (npermiso = 2) or (estado > 1)
{
   estado = not(estado)
}
```

CP: Entrada npermiso=2, contaminación=0, estado=3

# Diagrama de flujo de control






## Ejemplo 2

```
If nivel = Intermedio Then
    i=0
     While Estudiantes.count > 0
            If  Estudiantes.est[i].coord = 3  Then
                    esCoordClase = True
            End If
            i++
            Estudiantes.count  --
     Next
    If Not esCoordClase and i<5 Then
            msg("Error")
    End If
End If
```

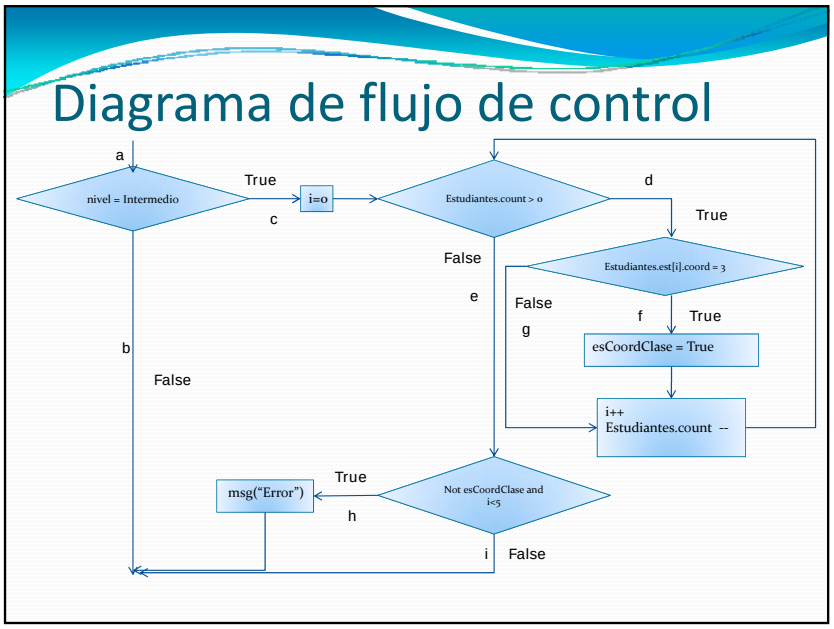
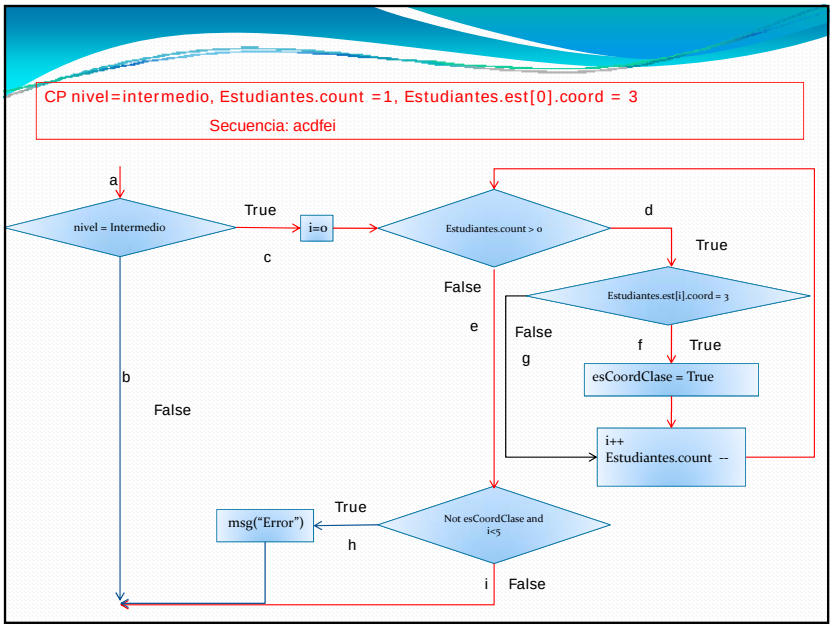
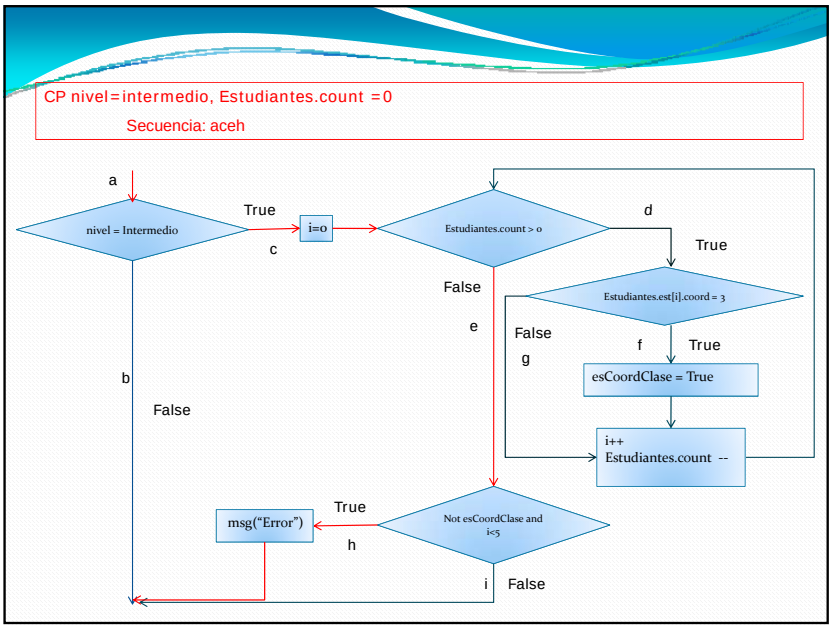


Diagrama de flujo de control
a
nivel = Intermedio
True
i=0
c
Estudiantes.count > 0
d
True
Estudiantes.est[i].coord = 3
False
e
False
g
f
True
esCoordClase = True
b
False
i++
Estudiantes.count --
True
msg("Error")
Not esCoordClase and i<5
h
i
False


CP nivel=intermedio, Estudiantes.count =1, Estudiantes.est[0].coord = 3
Secuencia: acdfei
a
nivel = Intermedio
True
i=0
c
Estudiantes.count > 0
d
True
Estudiantes.est[i].coord = 3
False
e
False
g
f
True
esCoordClase = True
b
False
i++
Estudiantes.count --
True
msg("Error")
Not esCoordClase and i<5
h
i
False




# Data Flow Testing

- Se define como una técnica de prueba que observa como los distintos valores asociados a variables pueden afectar la ejecución de un programa.
- Presta atención a como se define una variable y como se utiliza la misma a lo largo del flujo del control.
- Expresa los cubrimientos del testing en términos de las asociaciones definición-uso del programa.

## Definiciones

- Una ***definición*** de una variable ocurre cuando se asigna algún valor a la misma, ya sea de forma explícita o implícita.
- Una variable en el lado derecho de una asignación es llamada un ***uso computacional*** o un ***c-uso.*** Existen casos particulares que veremos en detalle más adelante.
- Una variable en un predicado booleano que implique una bifurcación resulta en un par ***uso predicado*** o ***p-uso*** correspondiendo a las evaluaciones verdadero y falso del predicado. Existen casos particulares que veremos en detalle más adelante.

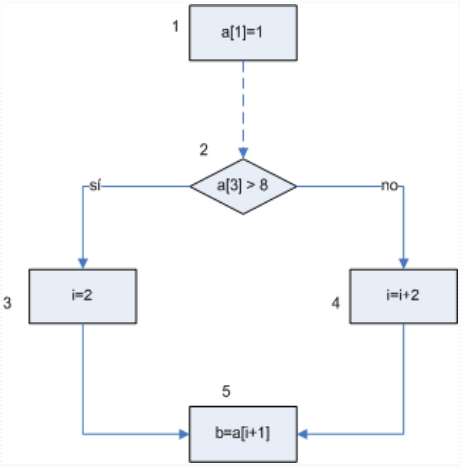
## Definiciones (2)

- Un camino $(i,n_1,n_2,\ldots,n_m,j)$ es un camino ***limpio-definición*** para x desde la salida de i hasta la entrada a j si no contiene una definición de x en los nodos de $n_1$ a $n_m$

## Cómo considerar Arrays en DTF ?

- Una asignación de **a[e]** siendo a un array, consiste en un c-uso de cada variable que aparece en la expresión **e** y una definición de la variable **a**. Ejemplo: a[5] = 0
- Una ocurrencia de **a[e]** que no sea una asignación, consiste en un uso de cada variable que aparece en la expresión **e** y un uso de la variable **a**. Ejemplo: b= a[8] + 1
- No distinguimos las definiciones y usos a nivel de las posiciones del array, debido a la imposibilidad de determinar a que posición se corresponde una expresión cuando es utilizada como índice.

## Ejemplo:



## Ejemplo (2)

- definición de **a**: Nodo 1
- c-uso de **a**: Nodo 5
- p-uso de **a**: Nodo 2

- Si consideramos las definiciones y usos a nivel de las posiciones del array dependemos del valor que tome la variable **i**.

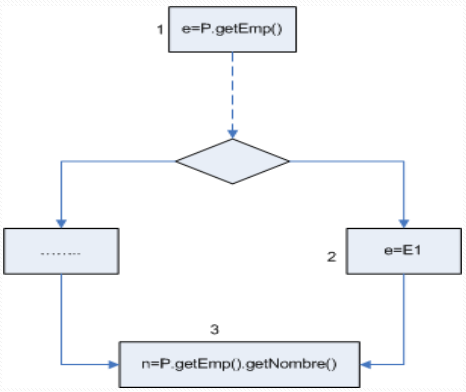
Consideramos definiciones y usos a nivel de la variable **a**.

## Cómo considerar punteros en DTF?

### Ejemplo:

P variable del tipo Persona
P tiene un pseudo atributo
"emp" de tipo Empresa

e variable del tipo Empresa



## Cómo considerar punteros en DTF? (2)

- Con la utilización de alias el uso de un mismo sector de memoria puede variar dependiendo del camino del grafo de flujo de control que se recorre, debido al empleo de variables de distinto nombre para referenciar a la misma locación de memoria.

## Ejemplo (2)

- definición de **e**: Nodo 1 y 2
- Si consideramos el uso de alias, existe un c-uso del pseudo-atributo de **P** en el nodo 3, que coincide con **e** sólo si se toma el camino A.

No tenemos en cuenta la utilización de alias debido a las dificultades planteadas anteriormente, con lo cual sólo consideramos que se corresponden las definiciones y usos de variables del mismo nombre.

## Criterios para sentencias de código

- v = expression:
  Un c-uso para cada variable en "expression" y una definición de v
- read($v_1$, $v_2$, . . . $v_n$):
  Una definición de cada $v_1$ . . . $v_n$
- write($v_1$, $v_2$, . . . $v_n$):
  Un c-uso de cada $v_1$ . . . $v_n$
- while B do S:
  Un p-uso por cada variable en la expresión booleana B

## Criterios para sentencias de código (2)

for (v=e1, v<= e2, v++):
Una definición de v, un c-uso por cada variable en e1, un p-uso de v, un p-uso por cada variable en e2 y un c-uso de v.

if B then $S_1$ else $S_2$:
Se considera un p-uso por cada variable en la expresión B, $S_1$ y $S_2$ se clasifican dependiendo de su composición.

## Criterios para sentencias de código (3)

- case $e_1$
  $S_1$:
  $S_2$:

  Un p-uso por cada variable en la expresión $e_1$, $S_1$ y $S_2$ se clasifican dependiendo de su composición.
- return $v_1$:

  Un c-uso de la variable $v_1$.
- return $v_1 > 0$:

  Un c-uso de la variable $v_1$.

## Data Flow Testing en Orientación a Objetos

- La unidad básica de testeo es la clase
- Puede ser aplicado tanto para el testeo de métodos individuales pertenecientes a una clase, como para métodos que interactúan con otros métodos de la misma.
- La interacción puede deberse a la cadena de llamadas provocada por la invocación de un método en particular, como a la ejecución de secuencias arbitrarias de métodos por parte de un usuario de la clase.

## Data Flow Testing en Orientación a Objetos (2)

El testeo de una clase puede realizarse en tres niveles diferentes:

- Intra-método
- Inter-método
- Intra-clase

## Data Flow Testing en Orientación a Objetos (3)

- Intra-método:

Corresponde al testeo de los métodos de la clase en forma individual, sin interacción con otros métodos. Este nivel es equivalente a pruebas unitarias de procedimientos individuales en programación estructurada.

## Data Flow Testing en Orientación a Objetos (4)

- Inter-método:

  Corresponde al testeo de métodos públicos junto con todos los métodos que son alcanzables desde él (es decir, que pertenecen a la secuencia de llamadas que dicho método dispara). Este nivel es equivalente a pruebas de integración de procedimientos en programación estructurada.

## Data Flow Testing en Orientación a Objetos (5)

- Intra-clase:

  Corresponde al testeo de las interacciones de los métodos públicos de una clase cuando son invocados en secuencias arbitrarias. Como existe un número infinito de combinaciones solo es posible testear un subconjunto de las mismas.

## Data Flow Testing en Orientación a Objetos (6)

En relación a los niveles presentados anteriormente se identifican tres tipos de pares definición-uso a ser testeados:

- Pares intra-método
- Pares inter-método
- Pares intra-clase

## Data Flow Testing en Orientación a Objetos (7)

- Pares intra-método:

  Son los que tienen lugar en métodos individuales, y testean el flujo de datos limitado a dichos métodos. Tanto la definición como el uso deben pertenecer al método bajo prueba.

## Data Flow Testing en Orientación a Objetos (8)

- Pares inter-método:

  Ocurren cuando interactúan métodos en el contexto de la invocación de un único método público. Son pares donde la definición pertenece a un método y el correspondiente uso se sitúa en un método llamado o en un método llamador (tanto llamadas directas como indirectas) en la misma cadena de invocaciones. El alcance del par definición-uso sobrepasa la frontera de algún método.

## Data Flow Testing en Orientación a Objetos (9)

- Pares intra-clase:

  Tienen lugar en el contexto de invocaciones a secuencias de métodos públicos. El alcance del par definición-uso sobrepasa la frontera de al menos dos métodos públicos de la clase

## Criterio para el testeo a nivel de intra-método

- Necesitamos establecer un criterio para las variables utilizadas en llamadas a otros métodos.
- Si la variable es sólo de entrada al método se considera un uso.
- Si la variable es sólo de salida se considera una definición.
- Si la variable es de entrada-salida se considera una definición y un uso.

## Criterio para el testeo a nivel de intra-método aplicado en Java

- Como en Java el pasaje de parámetros es realizado por valor, no existen parámetros de salida. Por lo tanto no se realizan definiciones de variables en las llamadas a otros métodos.

No aplica para arrays

## Cómo considerar atributos de la clase en intra-método ?

- Al testear un método **a** que llama a un método **b** de la misma clase se deben tener en cuenta las definiciones y usos de los atributos de la clase en el método **b**.
- Se asocia definición y/o uso del atributo en el nodo de la llamada al método **b**.
- El uso se clasifica en c-uso o p-uso según el contexto de la llamada.

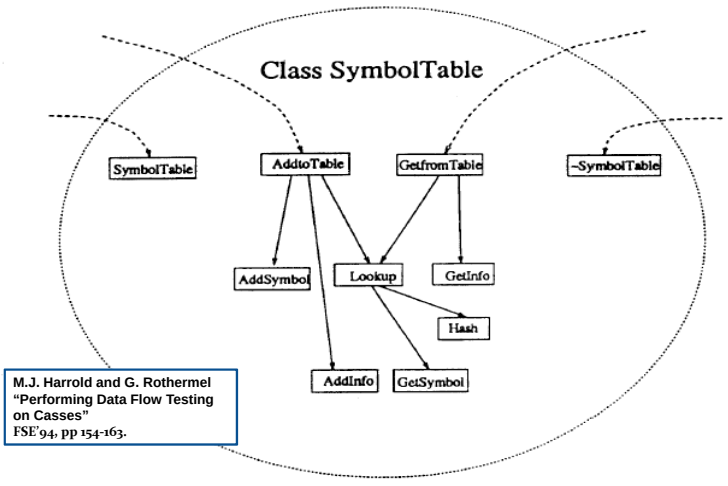
## Criterio para el testeo a nivel de inter-método

- Es necesario fijar el criterio para testear la integración de los métodos que se invocan de forma de directa o indirecta.
- Debemos contar con la información de las definiciones y usos de las variables involucradas en las llamadas a otros métodos de la secuencia de llamadas. Entendemos por variables involucradas a aquellas que se definen en el método testeado y se usan en otro método, así como las que se usan en el método testeado siendo definidas en otro método.

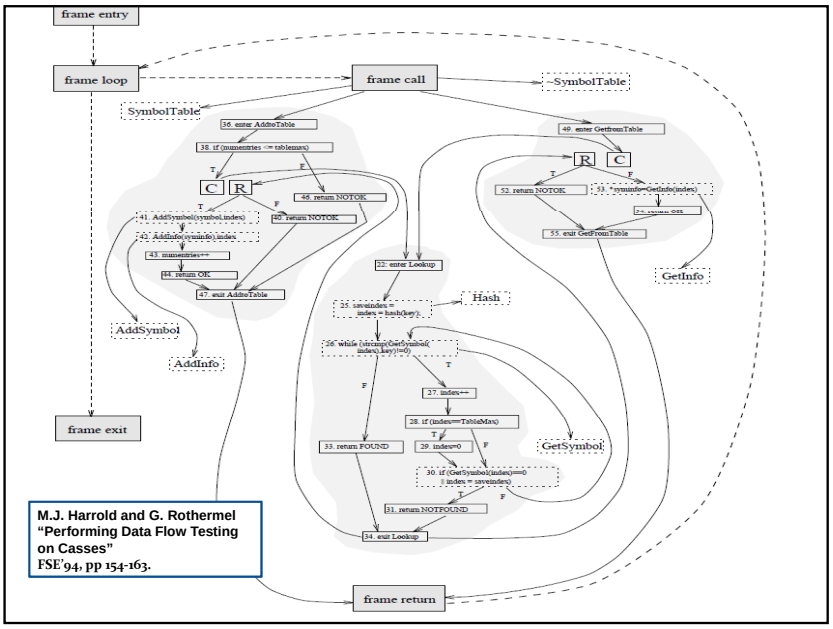
# Criterio para el testeo a nivel de inter-método (2)

- Tenemos que registrar la información del flujo de datos de todos los métodos que son accesibles desde fuera de la clase y los métodos que estos invocan de forma directa o indirecta. Se utiliza la información para testear las posibles interacciones entre los métodos.
- Con el propósito de registrar la información utilizamos una variante del Grafo de Flujo de Control de la Clase (CCFG) como representación gráfica, que denominamos CFG'.

# Grafo de llamadas




M.J. Harrold and G. Rothermel "Performing Data Flow Testing on Casses" FSE'94, pp 154-163.




## Pasos para la construcción del CFG'

- Salida: El CFG'
- Paso 1: Construir el grafo de llamadas
- Paso 2: Reemplazar cada nodo de llamada por el correspondiente grafo de flujo de control
- Paso 3: Reemplazar las llamadas a métodos por nodos de call y return
- Paso 4: Retornar el CFG'

# Algoritmo de construcción del CFG'

- Entradas: C (la clase a testear), y las clases accedidas desde la misma
- Salida: G (el CFG' )
- Paso 1: G = Grafo de llamadas
- Paso 2: Para todo método M en G hacer
  Reemplazar el nodo de llamada de M en G por el grafo de flujo de control de M
  Actualizar las aristas de forma apropiada
  Fin Para Todo
- Paso 3: Para todo nodo de llamada S en G, que represente la invocación a un método M hacer
  Reemplazar S por nodos de call y return
  Actualizar las aristas de forma apropiada
  Fin Para Todo
- Paso 4: Retornar G

# Criterio de detención a la extensión del CFG'

- Es necesario establecer un criterio de parada que determine el nivel de profundidad del grafo.
- El criterio es detener la extensión del grafo en el nivel 2 de profundidad.
- Consideramos nivel de profundidad 1 cuando estamos posicionados en un método llamado directamente por el método testeado. Generalizando se considera que un método pertenece al nivel n+1 cuando es invocado directamente por un método del nivel n.

## Cómo considerar invocaciones a objetos en predicados booleanos?

- If (lista.largo() > 0 )…
  Resp: p-uso de lista
- If (lista.algo(a,b,c) > 0 )…
  Resp: p-uso de lista
  a - in : p-uso
  b - inout : p-uso y definición
  c - out : definición

## Cómo considerar atributos en un método de la misma clase invocado por el método a testear?

- Al testear un método **a** que llama a un método **b** de la misma clase que utiliza atributos de la clase, consideramos las definiciones y usos de dichos atributos.

## Cómo considerar atributos en un método de otra clase invocado por el método a testear?

- Al testear un método de la clase **A** que llama a un método de la clase **B** que utiliza un atributo **b** de **B**, no consideramos las definiciones y usos de **b.**

## Cómo considerar variables locales en un método invocado por el método a testear?

- Al testear un método **a** que llama a un método **b** que utiliza variables locales, no consideramos las definiciones y usos de dichas variables locales.

## Cómo considerar parámetros en el método a testear?

- En la firma de cada método a testear consideramos definiciones de todos los parámetros de entrada.

  Esto aplica para intra-método y para el primer método de la cadena de llamadas de inter-método (es decir, el método a testear).

## Cómo considerar atributos en el método a testear?

- En el nodo inicial del CFG de cada método a testear considero definiciones de todos los atributos de la clase que son utilizados en el método.

  Esto aplica para intra-método y para el primer método de la cadena de llamadas de inter-método (es decir, el método a testear).

## Cómo considerar Try - Catch ?

- Al testear un método **a** que contiene **try-catch** sólo consideramos el código contenido en el bloque **try**.

## Inter-método aplicado en Java

- Al testar un método **a** que llama a un método **b**, cada parámetro **x** que el método **b** recibe se considera de la siguiente forma en **b**:
  - Se consideran todos los usos de la variable **x** previos a la primer definición de **x**.
  - No se consideran las definiciones de **x**.
  - En caso de tener usos de **x** posteriores a una posible definición de **x** (p.e: la definición está condicionada a un IF) consideramos dichos usos de **x**.

No aplica para arrays

## Arrays en Inter-método

- Al testar un método **a** que llama a un método **b**, cada parámetro **x** de tipo array que el método **b** recibe se considera de la siguiente forma en **b**:
  - Se consideran todas las definiciones de **x**.
  - Se consideran todos los usos de **x**.

## Anomalías en el flujo de datos

- Son defectos en el código producidos por un flujo de datos incorrecto. Es posible detectarlos durante la etapa de diseño de los casos.
- Pueden deberse, por ejemplo, a equivocaciones en los nombres de las variables, omisión de sentencias, y uso incorrecto de parámetros.

Ejemplos:

- dd : definiciones consecutivas
- d- : definición sin posterior uso
- -u : uso sin previa definición

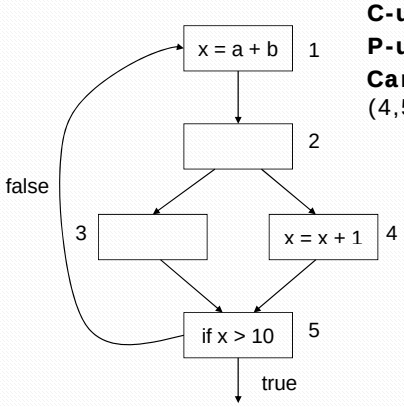
## Todos los usos

- Se ejecuta para cada definición **al menos un** camino libre-definición que lleve a **todos** los c-usos y p-usos correspondientes

## Pasos a seguir:

- Hallar las definiciones y usos de las variables.
- Hallar los caminos libre definición de las variables.
- Determinar los casos de prueba (valores de entrada y resultados esperados) para cumplir con el criterio.
- Ejecutar los casos de prueba
- Para los casos que fallan, buscar el defecto.

## Ejemplo 1

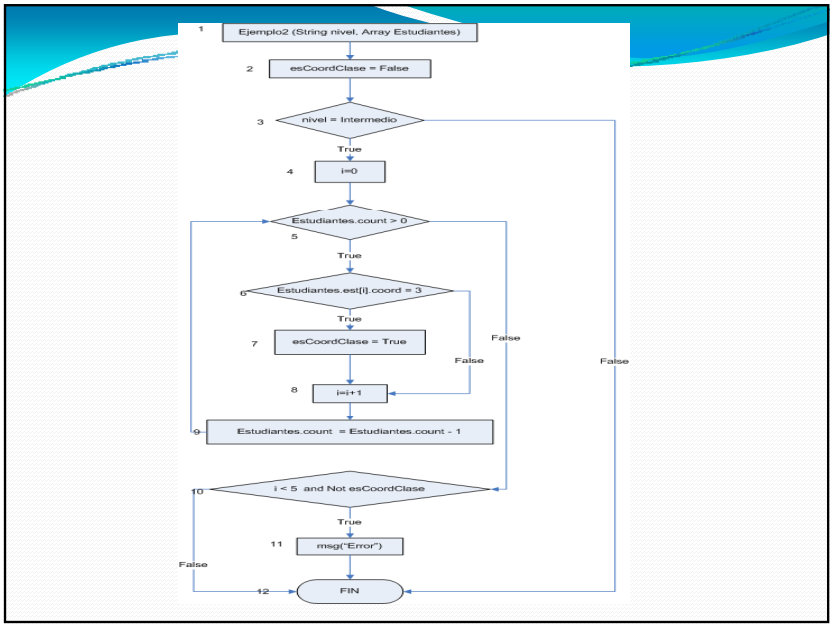



**Definiciones de x:** Nodos 1 y 4
**C-uso de x:** Nodo 4
**P-uso de x:** Nodo 5
**Camino libre-def para x:** (1,2,4) , (4,5) y (1,2,3,5)

**CP:** a=1, b=3
a=8, b=3

## Ejemplo 2

```
Ejemplo2 (String nivel, Array Estudiantes)
    esCoordClase = False
    If nivel = Intermedio Then
        i=0
        While Estudiantes.count > 0
              If  Estudiantes.est[i].coord = 3  Then
                  esCoordClase = True
              End If
              i=i+1
              Estudiantes.count  = Estudiantes.count - 1
        Next
        If i < 5  and Not esCoordClase Then
            msg("Error")
        End If
    End If
```




# Hallamos definiciones y usos

- **Definiciones de i:** Nodos 4 y 8
- **C-uso de i:** Nodo 8
- **P-uso de i:** Nodo 6 y 10

- **Camino libre-def de i:** (4,5,6,7,8),(4,5,10),(8,9,5,6),(8,9,5,6,7,8), (8,9,5,10), (4,5,6,8)

- **CP:** {nivel=intermedio, Estudiantes.count=1, Estudiantes.est[0].coord =3},
  {nivel=intermedio, Estudiantes.count=1, Estudiantes.est[0].coord =2}
  {nivel=intermedio, Estudiantes.count=0}
  {nivel=intermedio, Estudiantes.count=2, Estudiantes.est[0].coord =3, Estudiantes.est[1].coord =3}
  {nivel=intermedio, Estudiantes.count=2, Estudiantes.est[0].coord =3, Estudiantes.est[1].coord =2}
  {nivel=intermedio, Estudiantes.count=1, Estudiantes.est[0].coord =3}

## Hallamos definiciones y usos (2)

- **Definiciones de esCoordClase :** Nodos 2 y 7
- **C-uso de esCoordClase:** Sin c-uso
- **P-uso de esCoordClase:** Nodo 10

- **Camino libre-def de esCoordClase:** (2,3,4,5,10), (7,8,9,5,10)

- **CP:** {nivel=intermedio, Estudiantes.count=0}
  {nivel=intermedio, Estudiantes.count=1, Estudiantes.est[0].coord =3}

## Hallamos definiciones y usos (3)

- **Definiciones de nivel:** Nodo 1
- **C-uso de nivel:** Sin c-uso
- **P-uso de nivel:** Nodo 3

- **Camino libre-definición de nivel:** (1,2,3)

- **CP:** { nivel = Intermedio}, {nivel = Alto }

# Hallamos definiciones y usos (4)

- **Definiciones de Estudiantes:** Nodo 1 y 9
- **C-uso de Estudiantes :** Nodo 9
- **P-uso de Estudiantes :** Nodo 5 y 6

- **Camino libre-definición de Estudiantes**: (1,2,3,4,5,6,7,8,9), (9,5,6,7,8,9), (1,2,3,4,5,10), (1,2,3,4,5,6,8), (9,5,10), (9,5,6,8)

- **CP:** {nivel=intermedio, Estudiantes.count=1, Estudiantes.est[0].coord =3},
  {nivel=intermedio, Estudiantes.count=0},
  {nivel=intermedio, Estudiantes.count=1, Estudiantes.est[0].coord =2},
  {nivel=intermedio, Estudiantes.count=2, Estudiantes.est[0].coord =3,
  Estudiantes.est[1].coord =3},
  {nivel=intermedio, Estudiantes.count=2, Estudiantes.est[0].coord =3,
  Estudiantes.est[1].coord =2},
  {nivel=intermedio, Estudiantes.count=1, Estudiantes.est[0].coord =3}

# Bibliografía


- "Performing Data Flow Testing on Classes "
  Mary Jean Harrold and Gregg Rothermel
  Foundations of Softw. Eng. December 1994, pages 154-163
- "An Applicable Family of Data Flow Testing Criteria"
  Phyllis G. Frankl and Elaine J. Weyuker
  IEEE Transactions on Software Engineering, Vol 14. Nro. 10, October 1988
- "Interprocedural data flow testing"
  Mary Jean Harrold and Mary Lou Soffa
  ACM SIGSOFT '89 third symposium on Software testing, analysis, and verification, pages 158-167
- "Employing data flow testing on object-oriented classes"
  B.-Y.Tsai, SStobart and N.Parrington
  IEE Proc-Sofiw., Vol. 148, No. 2, April 200I

# Preguntas?

## Registro de Defectos

**Sistema De Registro De Defectos**

| | | | | | |
|---|---|---|---|---|---|
| Nombre | | Técnica | | Software | |
| Usuario | | Fecha de Inicio | | Fecha de Fin | |
| | | Tiempo de Diseño de Casos | | | |

Nombre: Nombre de la experiencia, en este caso Experiencia 0.
Usuario: Nombre de quién ejecuta la experiencia.
Técnica: (Sentencias, DTF)
Fecha de Inicio: Fecha de inicio de la experiencia. (22/08/09)
Tiempo de Diseño de Casos: Tiempo empleado en el diseño de los casos.
Software: Corresponde al nombre del Programa o producto a testear.
Fecha de Fin: Fecha de finalización de la experiencia.

# Registro de Defectos (2)

| Línea | Descripción | Archivo | Tiempo Detección | Estructura | Línea Estructura |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

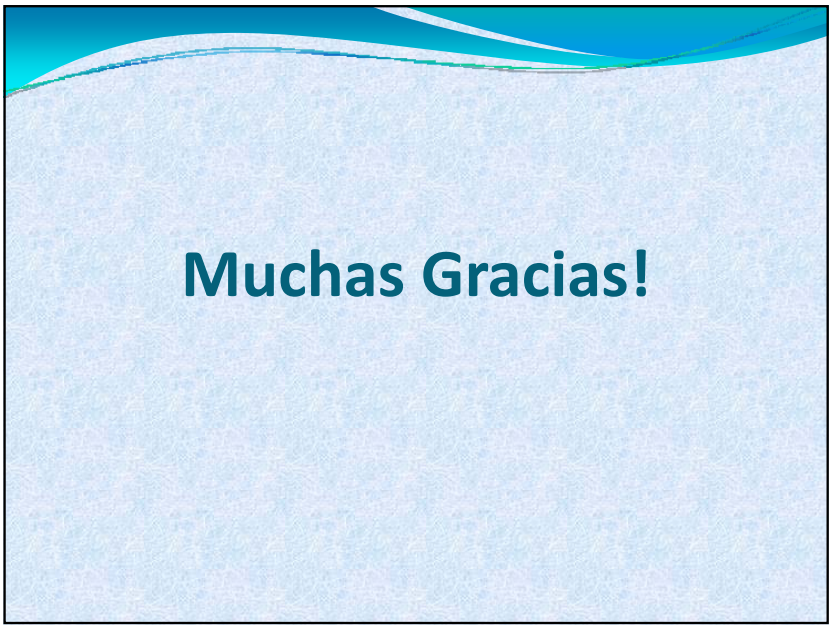
Línea: Línea del código donde se encuentra el defecto.
Descripción: Detalle del defecto detectado.
Archivo: Nombre de la clase que contiene el defecto.
Tiempo de Detección: Tiempo transcurrido entre el inicio de la búsqueda y el momento de detección del defecto.
Estructura: Se refiere a la estructura dentro de la cual se encuentra el defecto. Por ejemplo: IF, FOR, WHILE, Método.
Línea Estructura: Se refiere a la línea de comienzo de la estructura.

# Descripción de Defectos

- Describir el defecto en sí mismo, y no explicar el caso de prueba generado.
- Detallar una posible solución que corrija el defecto encontrado.

Ejemplo de descripciones incorrectas:

- La condición del IF es incorrecta.
- Retorna una cantidad mayor a la esperada.


Muchas Gracias!

# Apéndice B

# Guía para Técnica de Caja Blanca

A continuación se presentan las fases que intervienen en el proceso de verificación dinámica con caja blanca. El proceso comprende 3 fases: Preparación, Diseño y Ejecución.

**VER_DIN_CB0**
**Script del proceso de verificación dinámica con caja blanca**

| Paso | Actividades | Descripción |
|---|---|---|
| 1 | Preparación | ▪ Actividades de preparación para la verificación. |
| 2 | Diseño | ▪ Diseñar los casos de prueba que cumplan con el criterio de la técnica solicitada.<br>▪ Implementar los casos en JUnit.<br>▪ Registrar los datos requeridos de la fase. |
| 3 | Ejecución | ▪ Ejecutar los casos de prueba diseñados.<br>▪ Buscar los defectos asociados a los casos que fallan.<br>▪ Registrar los datos requeridos de la fase. |

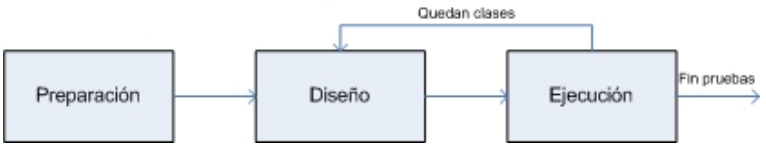



Figura B.1: Proceso de Pruebas

**Fase de Preparación:** Realizar un chequeo inicial garantizando que se puede llevar a cabo la verificación requerida.

**VER_DIN_CB0**
**Script para la fase de Preparación**

| Paso | Actividades | Descripción |
|---|---|---|
| 1 | Archivos | ▪ Verificar que se cuenta con los archivos fuentes a probar.<br>▪ Verificar que se cuenta con el Diagrama de Clases.<br>▪ Verificar que se cuenta con el javadoc de las clases a probar.<br>▪ Verificar que se cuenta con la planilla de registro de defectos y de tiempos. |

**Fase de Diseño:** Realizar el diseño de los casos de prueba cumpliendo el criterio de la técnica de verificación a aplicar. Se deben diseñar los casos de forma bottom-up.

**VER_DIN_CB0**
**Script para la fase de Diseño**

| Paso | Actividades | Descripción |
|---|---|---|
| 1 | Definir el Conjunto de Datos | ▪ Registre el tiempo de comienzo de esta actividad.<br>▪ Para cada método definir el conjunto de valores de entrada que cumplen con el criterio establecido. |
| 2 | Definir las salidas esperadas | ▪ Definir la salida esperada (o comportamiento esperado) para cada elemento del Conjunto de Datos y así conformar los casos de prueba. |
| 3 | Depuración | ▪ Eliminar los casos de prueba que no puedan ser ejecutados (caminos imposibles, etc.). Comprobar si aún se cumple con el criterio. Caso que no se cumpla volver al paso 1 buscando cumplir con el criterio. |
| 4 | Codificación de los casos de prueba en JUnit | ▪ Codificar todos los casos de prueba diseñados en JUnit. |
| 5 | Registro de fin de diseño | ▪ Registre el tiempo de fin del diseño. |

NOTA 1: En caso de pausas durante el diseño estas deben ser restadas del tiempo total. Debido a esto se ingresará el tiempo total de las pausas durante el diseño.

NOTA 2: Si durante la fase de diseño se encuentra algún defecto este debe ser registrado tal cual se explica en la fase de ejecución. En estos casos el tiempo de detección del defecto es 0 (cero).

**Fase de Ejecución:** Realizar la ejecución de los casos de prueba diseñados. Los casos se deben ejecutar bottom-up (igual de como han sido diseñados).

**VER_DIN_CB0**
**Script para la fase de Ejecución**

| Paso | Actividades | Descripción |
|---|---|---|
| 1 | Ejecutar caso de prueba | ▪ Ejecutar los casos de prueba |
| 2 | Analizar la salida obtenida | ▪ En caso que ningún caso haya fallado se termina la fase. |
| 3 | Encontrar defectos | ▪ Mientras existan casos de prueba que fallaron:<br>• Escoger un caso de prueba que falla.<br>• Inicializar el tiempo de búsqueda del defecto.<br>• Buscar en el programa el defecto que causa la falla. Se puede usar una herramienta de debug y ejecutar el caso de prueba.<br>• Detener el tiempo de búsqueda del defecto.<br>• Ejecutar el paso 4.<br>• Solicitar a un responsable del curso que solucione el defecto y haga entrega de una versión corregida.<br>• Volver a correr el caso de prueba para ver si obtiene bandera verde. En caso de continuar fallando volver al comienzo del paso 3. |

| | | |
|---|---|---|
| 4 | Registro de Defectos | ▪ Para cada defecto encontrado registrar los siguientes datos:<br>• Tiempo de detección del defecto.<br>• Descripción general del defecto. Es importante que la descripción sea clara y precisa. Es recomendable agregar una sugerencia de corrección.<br>• Nombre del archivo que contiene el defecto.<br>• Línea en la que se encuentra el defecto. En caso que el defecto no esté en una línea específica ingrese 0 (cero). Atención: si el archivo que se está verificando fue modificado con respecto al original (por haber corregido algún defecto), debe indicarse la línea del archivo original.<br>• Estructura asociada al defecto (ej.: IF, FOR, WHILE, nombre del método, etc.). En caso que el defecto no tenga línea asociada este campo se debe llenar obligatoriamente.<br>• Línea de comienzo de la estructura (obligatoria si se indica estructura asociada). |

NOTA 1: En caso que durante la búsqueda del defecto que causa la falla de cierto caso de prueba se encuentra otro defecto, este se debe registrar poniendo 0 (cero) en el tiempo de detección. Luego de registrarlo, se continúa con la búsqueda del defecto asociado al caso que falla.

NOTA 2: Los tiempos de detección que contienen segundos se redondean hacia arriba. Por lo tanto, los defectos que sean detectados en menos de 1 minuto deben ser registrados con tiempo de detección 1.

NOTA 3: Recordar que se solicita que todos los test JUnit den bandera verde para dar por terminado el testeo de una clase.

| | |
|---|---|
| Registro de Tiempo para el Main | ▪ Registrar en la herramienta Grillo la cantidad de tiempo invertido en la ejecución de los casos de prueba **solamente** de la clase main. |

# Apéndice C

# Encuesta a los Sujetos

En este Anexo se incluye el mail enviado a los sujetos como finalización del MT, junto con las respuestas brindadas por parte de los mismos a las preguntas solicitadas.

Estimado, Con este mail estamos dando por finalizado el MT.

Nos interesa contarles un poco los aportes que el MT le brindan al proyecto de grado de Silvana y Carmen y a la investigación que está realizando Diego, y por consiguiente, la importancia de su participación en el mismo.
Nuestro objetivo es comparar la efectividad y el costo de las técnicas Cubrimiento de Sentencias y Todos los Usos.
Para ello utilizamos la información de los defectos que detectaron, del tiempo insumido en la detección y en el diseño de los casos de prueba.

Los resultados que hemos obtenido son que:
No hay suficiente evidencia estadística para afirmar que una técnica sea más efectiva que la otra.
Respecto al costo (tiempo invertido) se observa que la técnica de Todos los Usos es algo más costosa que Cubrimiento de Sentencias, lo cual es esperado.

Nos gustaría, además, tener la opinión de cada uno de ustedes respecto al MT.

1. ¿Consideran que fue provechoso para ustedes participar del MT ?
2. ¿Qué opinión tienen de las clases brindadas en facultad?
3. ¿Qué opinión tienen del desarrollo del MT durante estas semanas?
4. ¿Consideran que aplicarían alguna de las técnicas en su ámbito laboral?
5. ¿Consideran que la carga de trabajo fue acorde?

Esperamos sus comentarios.

Saludos!
Silvana, Carmen y Diego

**1. ¿Consideran que fue provechoso para ustedes participar del MT?**

- $S_1$: Sí considero que fue provechoso. En particular me quedo con la idea de cómo funcionan las técnicas y para un futuro por ahí me sirve poder aplicarlas.

- $S_2$: La verdad que sí, laboralmente me abrió un poco el panorama del testing, y a nivel de facultad fue una materia que si bien se tuvo que trabajar duro en algún momento, no se me hizo pesada.

- $S_3$: Me pareció provechoso el curso para aprender las técnicas de testeo que sólo fueron mencionadas en Introducción a la Ingeniería de Software, pero no habíamos visto en la práctica, por lo tanto no conocíamos que tanto funcionaban y detectaban defectos realmente.

- $S_4$: Si, estuvo bueno, en particular nunca había aplicado formalmente ninguna técnica de verificación, me sirvió, por un lado reafirmé los conocimientos acerca del tema que venían de Introducción a la Ingeniería de Software, y por otro conocí y utilicé la herramienta JUnit por primera vez, me resultó una herramienta muy potente para verificación en Java.

- $S_5$: Me resultó provechoso el curso debido a que me interesaba profundizar un poco más en el área del Testing y en particular en el de caja blanca. Me hubiera gustado de repente utilizar alguna otra técnica, o alguna herramienta para testeo de cubrimientos.

- $S_6$: Desde el punto de vista académico, sí. Personalmente no estoy muy familiarizado con las técnicas de testing, y el poder aprender y aplicar algunas de ellas extiende mis conocimientos en un área que me resulta muy interesante.

- $S_7$: Sí. Si bien en algún momento realice pruebas de testeo sobre clases, nunca fue de forma tan minuciosa y detallada.

- $S_8$: Sí, fue provechoso el hecho de participar en este módulo de taller, por un lado porque quizás de otra manera no hubiera puesto en práctica estas técnicas de forma tan específica, ya que luego de lo visto en el curso de Ingeniería de Software no había profundizado más en el tema. Además considero que a pesar de que no sea el perfil preferido, la verificación es un área interesante y donde todavía hay mucho por investigar y generar.

- $S_9$: Si, fue provechoso, en mi caso aprendí a aplicar las técnicas de testing que sólo conocía teóricamente, y en el caso de Intra-Inter, ni siquiera sabía que existía.

- $S_{10}$: Para mi fue muy provechoso realizar el curso MT, me permitió profundizar mis conocimientos en técnicas de testing, cursar a distancia y completar créditos para la carrera. Con respecto al contenido del curso, me fue de interés pues se profundizó en técnicas de testing, cosa que no pude hacer en otras materia de facultad, por otro lado, aprendí a realizar practicas formales de testing en aplicaciones reales.

- $S_{11}$: Considero que en cierta medida sí, fue provechoso porque además de aprender los métodos fue una experiencia en la que solamente apliqué testing sin haber desarrollado. Además estuvo bueno haber aprendido JUnit, que me pareció una herramienta fácil de usar y útil.

- $S_{12}$: Sí, en mi caso particular nunca había hecho tareas de testing, por lo tanto, me sirvió para tener una experiencia aplicando las técnicas explicadas por ustedes. Además de las técnicas también me sirvió para conocer un poco la herramienta JUnit, que tampoco la había utilizado.

- $S_{13}$: Me parece que sí fue muy provechoso ya que aprendí a usar JUnit y pude tener más claras las técnicas de verificación aplicadas, pudiendo aplicarlas en futuros trabajos de ser necesario.

- $S_{14}$: Sí, aprendí sobre verificación que está interesante, obtuve créditos y el tiempo dedicado estuvo dentro de lo previsto.

**2. ¿Qué opinión tienen de las clases brindadas en Facultad?**

- $S_1$: Me pareció que las clases estuvieron super bien dictadas, nos dió para comprender bastante bien como es que se debería aplicar cada una de las técnicas dadas. A su vez, los ejercicios prácticos que se realizaron durante las clases ayudaron a comprenderlas mejor.

- $S_2$: Cansadoras de tan largas, pero mejor, así no teníamos que ir otros días, se entendió bien claro todo y con los ejercicios que hicimos no se precisa más.

- $S_3$: Las clases me parecieron correctas ya que vimos tanto el teórico, como lo necesario de práctico para poder realizar las técnicas sin mayor necesidad de preguntar para evacuar las dudas.

- $S_4$: Las clases fueron buenísimas, yo ya no me acordaba demasiado de lo visto en Introducción a la Ingeniería de Software, pero estuvo perfecto, la verdad que fue una exposición clarísima. Si algo malo tenían las clases era la extensión, pero bueno, son dos días, tampoco me quejo.

- $S_5$: Las clases brindadas en la facultad me resultaron más que correctas, se dió un buen paneo de los temas y fue muy provechoso la interacción entre docentes y estudiantes.

- $S_6$: Las clases estuvieron muy buenas. Fueron claras, ordenadas y re llevaderas. La jornada práctica me pareció un poco larga y cansadora, pero fue fundamental para evacuar muchas dudas. Los breaks y las idas a comer fueron importantísimas para que muchos estudiantes siguieran haciendo el MT.

- $S_7$: Fueron claras y estructuradas. Buena exposición del pizarrón y las ppts. Buenos ejemplos.

- $S_8$: Por el lado de las clases creo que fueron suficientes y bastante completas en el sentido de que sirvieron como componente central de la teoría y práctica, que después con lectura de las ppt alcanzaba para tener los

fundamentos de las técnicas. Se me ocurre que quizás se podría haber realizado en 3 jornadas de menor duración en lugar de 2, aunque eso también tendría influencia en el largo del taller.

- $S_9$: Fueron útiles ya que sin esas clases hubiera sido más difícil aplicar las técnicas y seguramente las hubiese aplicado con más errores, y está bueno de vez en cuando ir a comer y tomar algo con los docentes y demás compañeros.
- $S_{10}$: Considero que las clases brindadas en facultad fueron muy provechosas, ya que mostraron un buen panorama sobre las técnicas de testing que existen, sus características, aplicaciones, etc. Esto me permitió entender mucho más fácil el material ahorrando tiempo de estudio.
- $S_{11}$: Si bien fueron un poco extensas para mi gusto, considero que estuvieron bien dadas, fueron bastante claras. El material me fue útil y me pareció bastante completo. La instancia práctica fue fundamental para fijar conceptos y aclarar dudas.
- $S_{12}$: Me parecieron que estuvieron bien dictadas, que se cumplió con el objetivo de que entendamos las técnicas y su uso práctico. Como recomendación diría que fueron un poco extensas en el horario, tal vez sea mas provechoso dar 3 clases más cortas que 2 tan largas. Pero de todas formas alcanzó para entender lo que pretendía.
- $S_{13}$: Estuvieron muy bien, están muy bien dadas ya que son bien sintéticas y te permiten recordar las técnicas que fueron aprendidas en Introducción a la Ingeniería de Software. Con los ejemplos dados me pude manejar muy bien para realizar las técnicas sin problemas. La organización de las clases es tan llevadera que no te das cuenta de todas las horas que fueron.
- $S_{14}$: Muy buenas, creo que sólo una vez tuve que recurrir a las ppts. Se entendieron muy bien todos los conceptos presentados.

**3. ¿Qué opinión tienen del desarrollo del MT durante estas semanas?**

- $S_1$: Me parece que estuvo todo bastante bien. Silvana y Carmen respondieron a todas las preguntas rapidísimo.
- $S_2$: Seguimos un procedimiento claramente pautado al inicio y se siguió perfecto durante todo el MT, cuando solicité correcciones o tenía dudas siempre me encontré con las respuestas inmediatas por parte de ustedes.
- $S_3$: Me pareció bien como se desarrollo el MT.
- $S_4$: Impecable. Fue muy buena su disposición para responder a los diferentes inconvenientes que se fueron planteando. Les agradezco la buena onda y la buena disposición que tuvieron siempre para atender las preguntas, las correcciones y los pedidos que les hice. Gracias por eso.
- $S_5$: El desarrollo me pareció muy bien enfocado, incremental en carga en las semanas. El enfoque me parece correcto porque a medida que uno domina más una técnica desarrolla más trabajo. Me gustaría resaltar la

amplia disponibilidad y coordialidad de los docentes hacia nosotros, vale la pena mencionar aspectos como contestar mails sábados de noche o domingos de tarde los cuales son muy valorados para personas que nos cuesta un poco poder dedicar muchas horas fuera de los fines de semana.

- $S_6$: Las semanas fueron desproporcionadas. Las primeras dos semanas fueron lights, las dos últimas estuvieron acorde al tiempo acordado. Las semanas del medio fueron mucho más largas de lo que esperaba. A lo largo de las semanas la aplicación de las técnicas estuvo bien, pero estaría mejor si manejáramos alguna otra técnica más, para que no se vuelva tan monótono. Las respuestas de Silvana y Carmen fueron claras, concisas, y lo más importante, rápidas.

- $S_7$: Hubo que dedicarle su tiempo pero sin mayores dificultades. Excelente disposición de Silvana y Carmen para contestar las dudas planteadas.

- $S_8$: El desarrollo del MT fue bueno, estuvo bien por ejemplo el hecho de que cada uno eligiera en qué día de la semana entregar, y planificarlo desde el comienzo.

- $S_9$: Sobre el desarrollo lo que esta bueno destacar es la interacción semana a semana, que te obliga a mantener el ritmo y no perder el hilo de la asignatura, y también la flexibilidad que se tuvo con las entregas.

- $S_{10}$: El desarrollo del módulo fue según lo planteado en el curso, creo que está acorde a los créditos que ofrece.

- $S_{11}$: Sinceramente me pareció un poco aburrido, una vez que le agarrás la mano a las técnicas, su aplicación se convierte en un trabajo automático y tedioso. Por otro lado, la dinámica estuvo bien, nunca demoraron en contestarme dudas o en mandarme correcciones de errores.

- $S_{12}$: Me parece que fue realizado según lo pautado, ya sea en su duración y en la carga del trabajo. Además ante dudas, preguntas y/o correcciones, siempre tuvimos respuestas casi inmediatas por ustedes, en mi caso se veían incrementadas los fines de semana y siempre obtuve respuestas. Por lo cuál no tengo ninguna objeción al desarrollo del módulo.

- $S_{13}$: Tenía su carga horaria pero no es nada estresante. Lo mas aburridor para mi era la realización de los diagramas de flujos, que si bien son necesarios para facilitar encontrar los caminos, con mucha atención se podría llegar hacer mirando el código solamente. Esto reduciría considerablemente el tiempo de diseño, ya que en mi caso perdía muchas horas en los diagramas.

- $S_{14}$: Cumplió totalmente con las expectativas, siempre hubieron respuestas muy rápidas y concretas por parte de los responsables del MT.

**4. ¿Consideran que aplicarían alguna de las técnicas en su ámbito laboral?**

- $S_1$: Por ahora en el ámbito de trabajo donde estoy me parece que sería bastante tedioso utilizar estas técnicas, ya que trabajo desarrollando con

Genexus y a su vez utilizando patterns de Genexus lo que hace que se genere muchísimas líneas de código Java. En la versión de Genexus con que trabajo (9.0), no se cuenta con debuggers ni nada por el estilo, lo que lo hace mucho más dificultoso aún detectar las faltas.

- $S_2$: Sin duda lo sugeriría para que alguien más lo haga.
- $S_3$: La técnica de todos los usos, que fue la que me tocó a mi, me pareció que consumía demasiado tiempo, y los únicos defectos que detecté me pareció que podría haberlos detectado con casos de prueba hallados de forma más simple como cubrimiento de sentencias. Por lo tanto, si aplicara alguna técnica de las vistas, sería la de cubrimiento de sentencias ya que creo se detecta un similar número de defectos con menos tiempo en el diseño de los casos de prueba.
- $S_4$: No trabajo programando, en particular trabajo en el ambiente de redes, o sea que no tengo manera de aplicar éstas técnicas en mi trabajo. Sin embargo, opino que si trabajara en QA de Software, seguramente utilizaría alguna de éstas técnicas. No estoy seguro si en el ambiente empresarial es posible llevarlas a cabo formalmente. Igualmente, aún sin realizar las técnicas de manera estrictamente formal, sí me resulta muy útil, y muy buena cosa llevar adelante la verificación de alguna manera.
- $S_5$: Realmente creo que Inter e Intra no los utilizaría pero sí Sentencias o alguna otra técnica más rápida. También creo que podría ser posible usar Inter-Intra pero con análisis de código o alguna otra forma de acelerar el proceso. Además en funcionalidades muy complejas resulta un poco complicado utilizar intra. Otra alternativa es construir una herramienta que arme el grafo (no parece muy complicado) y armar los casos a partir de los grafos generados.
- $S_6$: En mi ámbito laboral seguro que no. Debido a la falta de recursos, considero que estamos muy desorganizados en todo lo que es desarrollo, y mucho más en testing. La validación y verificación que hacemos es muy vaga. Me parece que en alguna otra empresa con otros recursos la técnica podría servir, pero antes de aplicarla me informaría por otras técnicas existentes.
- $S_7$: Es difícil, tal vez para algún procedimiento en particular que estuviera dando muchos problemas. El tema es que los tiempos estimados en los proyectos no consideran muy bien el tiempo de desarrollo de pruebas unitarias. También depende del sistema, para sistemas complejos puede llegar a ser imposible.
- $S_8$: Con respecto a la aplicación de estas técnicas, considero que no las aplicaría hoy en el plano laboral, pero quizás por las características de los proyectos, y los apremios en cuanto a tiempo, etc. Considero que pueden llegar a ahorrar costo de mantenimiento y de búsqueda de defectos, siendo este costo muy importante en ciertas aplicaciones. Supongo que debe más que nada formar parte de la política o planificación de proyectos para que se dedique el tiempo necesario al testing de caja blanca. Lo que me parece en realidad es que antes tiene que por lo menos existir una real etapa de

verificación incorporada a los ciclos de los proyectos, que quizás hoy sigue faltando en algunos lugares.

- $S_9$: Las aplicaría en casos puntuales, ej: algún algoritmo complicado. No las aplicaría siempre debido al tiempo que consume.
- $S_{10}$: Considero que dichas técnicas son perfectamente aplicables en el desarrollo de software, siempre sujeto a ciertas condiciones. Considero que el ambiente propicio para la aplicación de las técnicas sería: disponibilidad de tiempo para su aplicación y testing de métodos no muy extensos. En mi caso, cuando me enfrentaba a métodos muy extensos, perdía un poco de interés y objetividad en el test.
- $S_{11}$: No creo, no me parecen técnicas viables para programas grandes, me parece que daría demasiado trabajo.
- $S_{12}$: Esto la verdad no lo tendría muy claro, si pienso en mi trabajo, con los tiempos que se manejan, me parece que no sería viable aplicar estas técnicas, no por un tema de dificultad, ni por los resultados obtenidos, sino que el problema sería el tiempo que implica aplicarlas. Por lo cuál habría que realizar algún estudio de costo/beneficio para decidir aplicarlas.
- $S_{13}$: Sí, pienso que si, que aplicaría la de Cubrimiento de Sentencias que es mucho mas fácil y ágil que la otra. Eso sí, no haría los diagramas de flujos.
- $S_{14}$: Yo no formo parte de un equipo de testing, de todos modos testeo como desarrollador pero no aplicando una técnica formalmente. Considero que en determinados casos aplicaría la técnica de cubrimiento de sentencias.

**5. ¿Consideran que la carga de trabajo fue acorde?**

- $S_1$: La verdad que me insumió más tiempo del que tenía previsto que me llevaría.
- $S_2$: En promedio me parece que sí , quizá algunos picos pero en promedio bien.
- $S_3$: Las horas dedicadas fueron solo un poco más de las estimadas al comienzo, con excepción de la semana 6 que particularmente me llevo más del doble de lo estimado, lo que hizo que me desfasara una semana.
- $S_4$: Si, estuvo bien. En particular solamente pasé muchas horas metido en el tema en la semana (o las semanas) que tuve problemas con la base, después estuvo bien.
- $S_5$: La carga de trabajo me pareció justa.
- $S_6$: En mi caso no, me llevó bastante más de lo que tenía pensado.
- $S_7$: Sí.
- $S_8$: En cuanto a la carga horaria creo que en algunas semanas superó bastante lo especulado, pero también al comienzo fueron inferiores las horas necesarias.

- $S_9$: Aunque en las últimas semanas, al menos en mi caso, me tomó más tiempo de lo esperado, eso compensaría el tiempo que me tomó en las primeras semanas que fue mucho menor, por lo que sí me parece que en total la carga de trabajo fue la esperada.
- $S_{10}$: Considero que el comienzo del curso fue tranquilo y con poca carga horaria. Desde mediados del curso en adelante siempre llegaba a una carga de 10hs, sobrepasándola en algunos casos.
- $S_{11}$: Sí, creo que si.
- $S_{12}$: Sí, por más que en las últimas semanas llevó mas de 10 horas semanales, me parece que queda compensado con el tiempo invertido en las primeras, que no alcanzaron a las 10 hs. Por lo tanto me parece correcto.
- $S_{13}$: Sí, la carga fue acorde a lo que se dijo en un principio y me parece que estuvo bien.
- $S_{14}$: Sí, yo tuve problemas pero por poder dedicarle sólo los fines de semana, si veo la carga total creo que fue acorde.