Comparando las Técnicas de Verificación Todos los Usos y Cubrimiento de Sentencias

Diego Vallespir[1], Carmen Bogado[1], Silvana Moreno[1], and Juliana Herbert[2]

[1] Instituto de Computación, Universidad de la República
Montevideo, Uruguay.
{dvallesp, smoreno}@fing.edu.uy, cmbogado@gmail.com
[2] Herbert Consulting
Porto Alegre, RS, Brazil.
juliana@herbertconsulting.com

Resumen. Este artículo presenta un experimento formal que compara el comportamiento de las técnicas de pruebas Cubrimiento de sentencias y Todos los usos. El diseño de este experimento es típico para un factor con dos alternativas. Un total de 21 sujetos realizan pruebas sobre un único programa pequeño. Los resultados indican que estadísticamente no se puede diferenciar entre la efectividad de ambas técnicas. Sin embargo, existe suficiente evidencia estadística como para afirmar que el costo de ejecutar Todos los usos es mayor que el de ejecutar Cubrimiento de sentencias; resultado que esperábamos encontrar.

Palabras clave: Pruebas, Programa, Técnica.

1 Introducción

Las pruebas unitarias de software están fuertemente establecidas en la industria. Sin embargo, aún no se conoce con certeza la efectividad y el costo de cada una. Esto hace que la decisión de qué técnica utilizar no sea trivial.

Transcurrieron varios años de investigación empírica en el tema y, sin embargo, no se lograron aún resultados definitivos.En *A look at 25 years of data* los autores examinan con detenimiento diferentes experimentos de pruebas de software llegando también a esta conclusión [1].

En el Instituto de Computación de la Universidad de la República estamos llevando adelante una serie de experimentos formales para obtener datos más precisos en este sentido. Actualmente culminaron 4 experimentos, este artículo describe uno de ellos.Los resultados de otros experimentos de esta serie se encuentran en [2,3].

El experimento que aquí se describe compara las técnicas de pruebas unitarias Todos los usos y Cubrimiento de sentencias para conocer su costo y efectividad.El costo es definido como el tiempo que lleva desarrollar los casos de prueba para cumplir con el cubrimiento exigido por la técnica. La efectividad se define como la cantidad de defectos encontrados al ejecutar la técnica dividido la cantidad de defectos totales del programa bajo prueba.

La sección 2 presenta los trabajos relacionados. La sección3 presenta el programa utilizado en el experimento. Las técnicas utilizadas se presentan en la sección 4. El diseño del experimento y su ejecución se encuentran en la sección 5.Los resultados del experimento y las conclusiones se presentan en la sección 6. En la sección 7 se encuentra el trabajo a futuro.

2 Trabajos relacionados

Diversos experimentos formales se ejecutaron para conocer la efectividad y/o costo de distintas técnicas de pruebas unitarias. En esta sección se presentan algunos experimentos que usan técnicas basadas en flujo de datos.

Weyuker, en 1990 presenta un experimento para conocer el costo de las técnicas de pruebas basadas en flujo de datos [4]. El costo lo mide como la cantidad de casos de prueba generados al aplicar la técnica.Se estudian las siguientes técnicas de prueba: Todos los c-usos, Todos los p-usos, Todos los usos y Todos los caminos definición-uso. Los resultados indican que la cantidad de casos de prueba necesaria para satisfacer estos criterios es muy inferior a la cota del peor caso calculada teóricamente en un trabajo anterior también de Weyuker [5].

Frank y Weiss presentan un estudio empírico donde comparan la efectividad de las técnicas Todos los usos y Cubrimiento de decisión [6]. Se usan 9 programas para los cuales se generan conjuntos de casos de prueba de forma aleatoria. En este experimento no intervienen verificadores humanos. Se toman los conjuntos de casos de prueba que cumplen con uno u otro criterio y se evalúa si cada uno de esos conjuntos detecta o no al menos un defecto. Como resultado obtienen, con un 99% de confianza, que el criterio Todos los usos es más efectivo en 5 de los 9 programas. En los otros 4 programas no se puede distinguir.

En 1994, Hutchins y otros publican un experimento cuyo objetivo es comparar la efectividad de una variante de la técnica Todos los caminos definición-uso y una variante de la técnica de Cubrimiento de decisión [7]. El experimento tiene características similares al experimento de Frank y Weiss. Sin embargo, en este experimento se usan tanto, casos de prueba generados al azar de forma automática, comoverificadores humanos. Como resultado se obtiene que no hay evidencia estadística que muestre que una técnica es más efectiva que la otra.

Li y otros realizan un experimento para comparar cuatro técnicas de pruebas unitarias: Mutantes, Todos los usos, Cubrimiento de pares de aristas y Cubrimiento de camino

principal [8]. Buscan conocer la efectividad (medida como la cantidad de defectos detectados sobre los defectos sembrados) y el costo (medido como la cantidad de casos de prueba que se necesitan generar para satisfacer cada criterio de prueba). Los casos fueron generados a mano con ayuda de herramientas para conocer el cubrimiento y otra para generar mutantes. El resultado es que la técnica de Mutantes descubre más defectos mientras que las otras tres son similares. Sorprendentemente (según los autores), la técnica de Mutantes fue la que menos casos de prueba requirió.

Uno de los puntos que consideramos débiles de algunos de estos experimentos es que miden el costo como los casos de prueba que se necesitan generar para cumplir con cierto criterio de prueba. Entendemos que el tiempo invertido en desarrollar esos casos es una medida más interesante para el costo.

3 El Programa

El programa utilizado en este experimento es simple y está escrito en Java. El mismo recibe un array de enteros como parámetro y lo devuelve sin elementos repetidos y ordenado de menor a mayor. Consta de dos clases (Ordenador y OrdenadorSinRep), una de ellas contiene 18 líneas de código sin comentarios y la otra contiene 19. 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 repetidos.

A continuación se presenta la firma, la especificación y el código fuente del método **ordenar** de la clase **Ordenador** y del método **ordenarSinRep** de la clase **OrdenadorSinRep**.

public static void ordenar(int[] a)
El método retorna el array ordenado de menor a mayor.
En caso de que el array sea nulo o vacío permanece incambiado.
a: parámetro de entrada que contiene los enteros a ordenar.

```
public static void ordenar (int[] a){
      for(int i=a.length-1; i>0; i--){
            int swapped = 0;
            int find = 0;
            for (int j=0; j<i; j++){
                  if (a[j] > a[j+1]){
                        int aux = a[j];
                        a[j+1] = a[j];
                        a[j] = aux;
            swapped=1;
```

```
                }
                }
                if (swapped == 0) {
                      return;
                }
          }
    }
```

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).

```
    public static int ordenarSinRep(int[] a){
          int cantEliminados = 0;
          Ordenador.ordenar(a);
          for(int i=0; i<a.length-1; i++){
                if (a[i] == a[i+1]) {
                      desplazar(a, i+1);
                      cantEliminados++;
                }
          }
          return cantEliminados;
    }


    private static void desplazar(int[] a, int i){
          for(int j=i; j<a.length-1; j++){
                a[j]=a[j+1];
          }
    }
```

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 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.

En este experimento los defectos en el código son clasificados según si el defecto puede causar una falla (Posible Falla PF) o no (No Falla NF). Los NF 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,B,C,D,E,F,G) y OrdenadorSinRep tiene 6 (a,b,c,d,e,f). 5 son PF (A,F,a,b,d) y 8 son NF (C,D,E,F,G,c,e,f).

La descripción detallada de cada uno de estos defectos se encuentra en [2].

4 Las Técnicas

Se utilizan dos técnicas de verificación, ambas de caja blanca. Cubrimiento de sentencias (CS) que se basa en el flujo de control y Todos los usos (TU) que se basa en el flujo de datos.

Para cumplir con la técnica de CS se debe asegurar que cada instrucción del código fuente se ejecuta al menos una vez en el conjunto de casos de prueba construido. Al ser esta técnica ampliamente conocida no se profundiza en la misma en este artículo.

La técnica TU expresa el cubrimiento del testing en términos de las asociaciones definición-uso del programa. Una *definición* de una variable ocurre cuando un valor es almacenado en la variable (*x:= 7*). Un *uso* de una variable sucede cuando se lee (o usa) el valor de esa variable. Esto puede ser tanto un p-uso como un c-uso. Un *p-uso* es el uso de una variable en una bifurcación del código (*if (x==7)*). Un *c-uso* es cuando el uso no es en una bifurcación. Por ejemplo, en *(x:= 7 + y)* se tiene una definición de *x* y un c-uso de *y*.

Un camino $i_1, i_2, \ldots, i_n$ es *libre-definición* para la variable *x* si *x* no es definida en los nodos intermedios $(i_2, \ldots, i_{n-1})$.Una definición *d* (de una variable *x*) *alcanza* un uso *u* (también de la variable *x*) si existe un camino libre-definición desde *d* hasta *u*.

TU requiere que se ejecute al menos un camino libre-definicióndesde cada definición *d* hasta cada uso alcanzable.Las definiciones clásicas de las técnicas basadas en flujo de datos y en particular TU se presentan en un artículo de Rapss y Weyuker [9].

En Orientación a Objetos la unidad básica de pruebas es la clase. Es necesario probar sus métodos de forma individual y en forma colectiva, de manera de probar las interacciones que se generan a través de las secuencias de llamadas provocadas por la invocación de un método en particular.TU puede ser aplicada tanto para las pruebas de métodos individuales pertenecientes a una clase, como para métodos que interactúan con otros métodos de la misma clase o de otras clases.

Las pruebas de una clase en TU pueden realizarse en dos niveles: Intra-método (Intra) e Inter-método (Inter). En **Intra** se considera para el cubrimiento de código solamente el método bajo prueba. Entonces, en este caso, al momento de desarrollar los casos de prueba, no se consideran los métodos que interactúancon el método bajo prueba. Por otro lado, en **Inter** sí se consideran para el cubrimiento de código a los métodos que interactúancon el método bajo prueba.

En relación a los niveles presentados anteriormente se identifican dos tipos de pares definición-uso a ser probados.Los **Pares Intra-método** son los que tienen lugar en métodos individuales, y prueban el flujo de datos limitado a dichos métodos. Tanto la definición como el uso pertenecen al método bajo prueba. Los **Pares Inter-método** ocurren cuando existe interacción entre métodos. Son pares donde la definición pertenece

a un método y el correspondiente uso se sitúa en otro método que pertenece a la cadena de invocaciones.

En la mayoría de la literatura que presenta técnicas basadas en el flujo de datos se brindan ejemplosque contienen variables simples como enteros o booleanos. Sin embargo, al momento deaplicar estas técnicas usando arrays o aún "peor" objetos, se deben definir ciertos criterios que normalmente no son tratados.

Establecer estos criterios es fundamental para conocer bajo qué condiciones se aplica la técnica. Distintas condiciones pueden ocasionardiferentes resultados en la efectividad y el costo de TU ya que en realidad son técnicas diferentes con el mismo nombre. Muchas de estas condiciones refieren a cómo se deben considerar los Pares Inter-método.Este experimento establece las condiciones, para la aplicación de la técnica TU, en base a lo propuesto en [10, 11, 12].

5 Diseño y Ejecución del Experimento

En esta sección se presentan tanto las consideraciones generales del diseño y la ejecución del experimento (subsección 5.1) como el proceso seguido por los sujetos al momento de realizar las pruebas sobre el programa (subsección 5.2).

5.1 Consideraciones Generales del Diseño y la Ejecución

Este experimento consta de dos experiencias. La primera se realiza con 10 sujetos y la segunda con 11. Estas experiencias son realizadas con 3 semanas de diferencia entre una y otra.

La **unidad experimental** es el programa de ordenamiento de array. Las**alternativas** son las técnicas a evaluar (CS y TU). El diseño del experimento que se utiliza es típico para experimentos de un factor con dos alternativas. La asignación de los sujetos a la técnica a utilizar es totalmente al azar. De forma de que el diseño sea lo más balanceado posible, para la primer experiencia se tiene igual cantidadde sujetos para ambas técnicas y para la segunda se tiene uno más para TU.

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.

Las hipótesis de este experimento también son las tradicionales en este caso. La hipótesis nula de efectividad, hipótesis que se quiererechazar, plantea que las medianas de la efectividad de las técnicas son iguales. La hipótesis nula de costo plantea que las

medianasde costo de las técnicas son iguales. Las hipótesis alternativas correspondientes simplemente indican que las medianasson diferentes.

Los 21 sujetos que participan en las experiencias tienen similares características, por lo que consideramos al grupo como homogéneo. Todos son estudiantes de la carrera Ingeniería en Computación de la Universidad de la República. Son estudiantes avanzados de la carrera ya que todos se encuentran en cuarto o quinto año.

Las dos experiencias se realizan en tres sesiones: una sesión inicial de aprendizaje, una sesión de capacitación, y una de ejecución individual.

La **sesión de aprendizaje** tiene como objetivo que cada sujeto aprenda JUnit, herramienta que usarán luego los sujetos para codificar los casos de prueba. Esta sesión comprende estudio individual por parte de los sujetos y la entrega de un pequeño ejercicio.

La **sesión de capacitación** tiene como objetivo que los sujetos aprendan las técnicas CS y TU. Para esto se dicta un curso teórico/práctico de 9 horas que se ejecuta durante un día. En este curso también se presenta a los sujetos la guía que deben seguir para hacer las pruebas y las planillas para registros de tiempo y defectos.

La **sesión de ejecución individual** es realizada también durante todo un día, con una diferencia de 7 días con la sesión de capacitación. En esta sesión los sujetos aplican de forma individual la técnica que les fue asignada, generando los casos de prueba necesarios y ejecutando los mismos. Para llevar a cabo el trabajose sigue la guía proporcionada en la capacitación, registrando los tiempos y defectos tal como se indica en la misma. Los sujetos, al finalizar la sesión, entregan: 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, etc).

5.2 Proceso Seguido por los Sujetos

Los sujetos, durante la sesión de ejecución individual, siguen la guía de verificación que se les presentó y entregó en la sesión de capacitación. Esta establece el mini-proceso que deben seguir durante la verificación.

La guía establece que ambas técnicas deben ser ejecutadas puramente de caja blanca. Esto quiere decir que no se deben generar casos de prueba de caja negra, luego observar el cubrimiento alcanzado con alguna herramienta adecuada y por último complementar con casos que faltan para alcanzar el cubrimiento. Entonces, cada sujeto, genera sus casos de prueba a partir del código fuente buscando cumplir con el cubrimiento exigido por la técnica que le fuera asignada (CU o TU). No se usan herramientas de cubrimiento de código durante el experimento.

La guía establece, que los sujetos que deben aplican TU, deben primero generar los casos basados en Intra-método para luego generar los casos Inter-método. No queda establecido ningún otro aspecto de cómo aplicar cada técnica, esto queda a elección de cada sujeto.

Por último, el mini-proceso establece cuáles son los datos que se deben recolectar. Los sujetos deben registrar: el tiempo total que les lleva desarrollar y codificar en JUnit los casos de prueba, cada uno de los defectos encontrados, y el tiempo que lleva detectar un defecto luego de provocada una falla.

6 Resultados y Discusión

Se cuenta con los siguientes datos de cada sujeto: cuáles son los defectos que el sujeto detectó, el tiempo total de diseño y codificación de los casos de prueba, y el tiempo de detección para cada defecto detectado. Por motivos de espacio solamente se presenta el análisis de la efectividad de las técnicas y el costo de las mismas (tiempo de diseño y codificación).

6.1 Efectividad de las Técnicas

La efectividad promedio obtenida usando la técnica Cubrimiento de sentencias fue de un 29,2% y su desviación estándar fue de un 10,8%. Todos los usos tuvo una efectividad un poco mayor. Su efectividad promedio fue de 34,3% y su desviación estándar fue de 7,2%.

Se aplican los test no paramétricos de Mann-Whitney y de Kruskal-Wallis para conocer si se puede afirmar que TU fue más efectiva, con validez estadística, que CS. La hipótesis nula plantea que las medianas de efectividad son iguales, la hipótesis alternativa plantea que son distintas:

$$H_0: \mu_{CS} = \mu_{TU}. \quad (1)$$

$$H_1: \mu_{CS} <> \mu_{TU}. \quad (2)$$

Ninguno de los test rechaza H_0 con $\alpha \leq 0,1$ (valor de α habitual con el que trabajamos en nuestros experimentos). En este sentido entendemos que se necesitan más observaciones (participan solamente 21 sujetos en este experimento) para poder concluir sobre la efectividad con validez estadística. También es razonable variar los programas utilizados en el experimento y que estos sean más complejos.

Mencionamos que anteriormente realizamos otros experimentos. Uno de ellos se realizó con el mismo programa que se utiliza en este experimento pero con otros sujetos y otras técnicas de verificación [2].En ese trabajo 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.** La tabla1 presentaqué tan efectivas fueron las técnicas respecto a cada defecto en este experimento. Esto se calcula dividiendo la cantidadde sujetos que encontró el defecto entre la cantidad total de sujetos.

Tabla 1. Porcentaje de detección de defectos

Técnica	Defectos												
	A	B	C	D	E	F	G	a	b	c	d	e	f
Sentencias	40	100	0	0	0	0	90	20	60	0	70	0	0
Todos los usos	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 experimentoson esos y además el defecto G. El resto de los defectos no se detecta. Entonces, se puede concluirlo mismo que en el experimento anterior.

Las técnicas dinámicas buscan defectos mediante la provocación de fallas. Por lo tanto, es bastanteesperada esta primer conclusión. Sin embargo, en este experimento los sujetos revisan el código en dos oportunidades. Primero revisan el código detenidamente para generar un conjunto de casos de prueba que cumpla con el cubrimiento exigido por la técnica asignada. Luego revisan el código para cada falla que arroja JUnit para encontrar el defecto correspondiente. Lo que muestra este experimento es que la atención de los sujetos está únicamente en la tarea que están realizando. Les es difícil observar si hay defectos mientras generan casos de prueba. También les es difícil detectar un defecto que no es el que están buscando.

Considerando el experimento anterior y el presentado en este artículo, se tienen los siguientes promedios de efectividad por técnica: Todos los usos 34,3%, Inspección individual 31%, Tablas de decisión 31%, Cubrimiento de sentencias 29,2%, Clases de equivalencia 27% y Criterio de condición múltiple 17%.

6.2Costo de las Técnicas

El costo es definido como el tiempo invertido en el diseño y codificación en JUnit de los casos de prueba. El costo promedio de la técnica TU fue de 328 minutos mientras que el costo promedio de CS fue de 133 minutos. Es decir, en promedio TU fue casi 2,5 veces más costosa que CS.

Tanto Mann-Whitney como Kruskal-Wallis rechazan la hipótesis nula que establece que los costos de las técnicas son iguales. Entonces, para este experimento, existe suficiente evidencia estadística para afirmar que TU es más costosa que CS. Resultado que esperábamos obtener por la propia complejidad intrínseca de cada una de las técnicas.

En lo que refiere al costo lo que es más interesante es que TU fue mucho mas costosa (2,5 veces más). Sin embargo, la ganancia en efectividad no fue tan importante.

7 Conclusiones

En este trabajo se presenta un experimento formal para comparar las técnicas CS y TU en lo que refiere a su costo y a su efectividad. El costo se mide como el tiempo utilizado por el sujeto para desarrollar los casos de prueba. La efectividad se define como la cantidad de defectos encontrados dividido la cantidad de defectos totales.

Se utiliza un único programa escrito en Java como unidad bajo prueba. Este programa tiene al menos dos problemas: es muy simple y tiene una densidad de defectos demasiado alta. Entonces, es importante realizar otros experimentos con programas más complejos y una menor densidad de defectos.

El diseño del experimento es típico de un factor con dos alternativas. En este caso se divide en dos experiencias, una con 10 sujetos (5 ejecutan CS y 5 ejecutan TU) y otra con 11 sujetos (5 ejecutan CS y 6 ejecutan TU).

Los sujetos son todos estudiantes avanzados de la Carrera Ingeniería en Computación de la Universidad de la República en Uruguay. Además, se hacen dos sesiones de nivelación y enseñanza; *sesión de aprendizaje* y *sesión de capacitación.*

Durante la ejecución los sujetos desarrollan los casos de prueba siguiendo una guía que se les presentó y entregó en la sesión de capacitación. Los casos de prueba se deben codificar en JUnit. La construcción de estos es puramente de caja blanca y no se usaron herramientas de cubrimiento de código; el cubrimiento debían asegurarlo por construcción de los casos de prueba.

Como resultado se obtiene que la técnica TU tiene una mayor efectividad que la técnica CS; 34,3% es la efectividad promedio de TU y 29,2% de CS. Sin embargo, los test estadísticos realizados no rechazan la hipótesis de igualdad en la efectividad de ambas técnicas.

Por otro lado, en este experimento encontramos que la técnica TU es notoriamente más costosa que la técnica CS; 328 minutos para TU y 133 minutos para CS es lo que llevó en promedio desarrollar los casos de prueba. Los test estadísticos aplicados muestran que en este experimento hay suficiente evidencia estadística para afirmar que TU es más costosa que CS.

Actualmente nos encontramos analizando datos de un nuevo experimento que hemos ejecutado y también analizando datos de forma cualitativa de experimentos anteriores. Este nuevo experimento es con un programa más real de 1820 líneas de código sin comentarios y 14 clases, también desarrollado en Java.

En un futuro próximo pensamos ejecutar experimentos con una mayor población de sujetos. Estamos también interesados en conocer el impacto de utilizar herramientas de cubrimiento de código tanto en el costo como en la efectividad de las técnicas de caja blanca.

Referencias

1. Moreno, A., Shull, F., Juristo, N., Vegas, S.: A look at 25 years of data. IEEESoftware 26(1) (Jan.-Feb. 2009) 15-17
2. Vallespir, D., Herbert, J.: Effectiveness and cost of verification techniques: Preliminary conclusions on five techniques. In: Computer Science (ENC), 2009 MexicanInternational Conference on. (2009) 264-271
3. Vallespir, D., Apa, C., De León, S., Robaina, R., Herbert, J.: Effectiveness of five verification techniques. In: Proceedings of the XXVIII International Conference ofthe Chilean Computer Society. (2009)
4. Weyuker, E.: The cost of data flow testing: An empirical study. IEEE Transactionson Software Engineering 16 (1990) 121-128
5. Weyuker, E.J.: The complexity of data flow criteria for test data selection. InformationProcessing Letters 19(2) (1984) 103-109
6. Frankl, P.G., Weiss, S.N.: An experimental comparison of the effectiveness ofbranch testing and data flow testing. IEEE Transactions on Software Engineering 19(8) (1993) 774-787
7. Hutchins, M., Foster, H., Goradia, T., Ostrand, T.: Experiments on the effectivenessof dataflow- and control-flow-based test adequacy criteria. In: SoftwareEngineering, 1994. Proceedings. ICSE-16., 16th International Conference on. (16-21 1994) 191-200
8. Li, N., Praphamontripong, U., Offutt, J.: An experimental comparison of four unittest criteria: Mutation, edge-pair, all-uses and prime path coverage. In: SoftwareTesting, Verification and Validation Workshops, 2009. ICSTW '09. InternationalConference on. (1-4 2009) 220-229
9. Rapps, S., Weyuker, E.J.: Data flow analysis techniques for test data selection. In:ICSE '82: Proceedings of the 6th international conference on Software engineering,Los Alamitos, CA, USA, IEEE Computer Society Press (1982) 272-278
10. Harrold, M.J., Rothermel, G.: Performing data flow testing on classes. SIGSOFTSoftw. Eng. Notes 19(5) (1994) 154-163
11. Harrold, M.J., Soffa, M.L.: Interprocedual data flow testing. In: TAV3: Proceedingsof the ACM SIGSOFT '89 third Symposium on Software Testing, Analysis, and Verification, New York, NY, USA, ACM (1989) 158-167
12. Frankl, P.G., Weyuker, E.J.: An applicable family of data flow testing criteria.IEEE Transactions on Software Engineering 14(10) (1988) 1483-1498