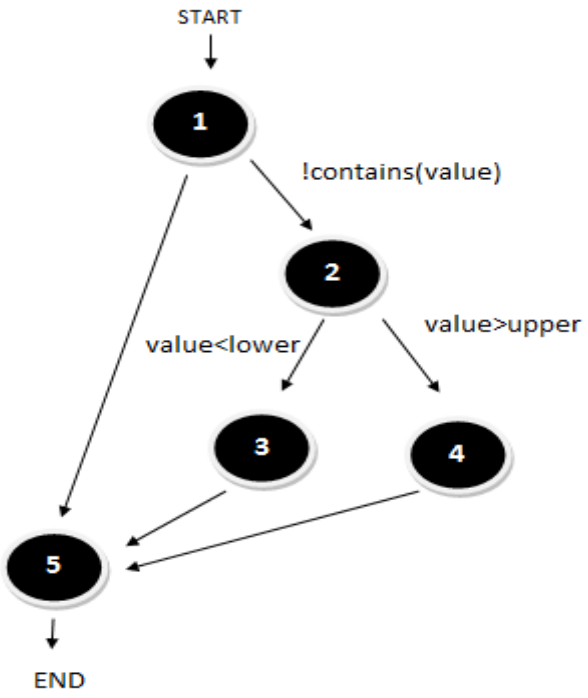
Lab. Report #3 – Code Coverage, Adequacy Criteria and Test Case Correlation

Group #:	16
Student Names:	Burcu İskender
	Kürşat Aktaş

1 INTRODUCTION

Bu assignmentta çeşitli metodlar üzerinde whitebox testi uyguladık.White box testi yaparken statement coverage, branch coverage,loop covarege ve condition covarege üzerinde çalıştık.Bunu uygularken de bir code coverage toolu olan EclEmmayı kullandık.Ancak not düşmek gerekirse EclEmma ' da istenen oranlar ayrıntılı olarak gösterilmiyordu.Biz de laboratuvar dersinde söylendiği gibi genel covarege oranlarıyla ilgilendik.Öncelikle code coverage uygulamasında tool bize oranlara gore yeşil,kırmızı ve sarı çıktılar Verdi(branch missed oranları).Test coveragei kavramak amacıyla 2. Laboratuvarda oluşturduğumuz test suiti kullandık. Daha sonra Range, DataUtilies ve DefaultKeyedValues ve bunların test classlarındaki coverage oranlarını alarak bu oranları yükseltmek amacıyla test planı oluşturduk ve buna bağlı olarak kod üzerinde iyileştirmeler yaptık.

2 MANUAL DATA-FLOW COVERAGE CALCULATIONS FOR `RANGE.CONSTRAIN(DOUBLE)` METHOD



node	defines	c-uses	p-uses
1	value,result,upper,lower	value	
2		value	
3	result	upper	value,upper
4	result	lower	value,lower
5		result	

node	dcu(v,i)	dpu(v,i)
1	dcu(upper,1)={3} dcu(lower,1)={4} dcu(value,1)={(1,2)} dcu(result,1)={5}	Dpu(value ,1)={(3,4)} Dpu(upper,1)={(3,4)} Dpu(lower,1)={(3,4)}
2	-	-
3	dcu(result,3)={5}	-
4	dcu(result,4)={5}	-
5	-	-
	Total number of def-clear c-use paths to cover:6	Total number of def-clear p-use paths to cover:4

Örnek olarak; aralıktaki bir value için sırasıyla 1 ve 6 nodelarına gidilecek.Value ve result değerleri tanımlanacak ve kullanılacak.

- Result definition ı node 1 de yapılacak ve ve node 1 de c-use olacak.
- Aynı şekilde value node 1 de tanımlanacak ve node 1 de c-use olacak.

Toplam 10 path gidilecek 2 olduğu için coverage oranı 2/10 = %20.

3 A DETAILED DESCRIPTION OF THE TESTING STRATEGY FOR UNIT TESTING

Unit testingde ,bir metodu test etmek için küçük test edilebilir birimlere ayırdık ve oluşturduğumuz birbirinden bağımsız test caselerle başka bir kod tarafından otomatik test edilmesini sağladık.Test etmek

istediğimiz kod segmentini diğer koddan ayırıp one göre test case oluşturduk.Her birim test edildikten sonra bütünü oluşturan parçaları unit test için kullanılabilir hale getirdik.

Unit testingin diğer test yöntemlerine göre avantajı olarak birimler bütünden bağımsız olarak test edildiği için kodda iyileştirme yapmayı kolaylaştırır.Yani kodun bozulmasından korkulduğu için kodda yenilik iyileştirme yapmaktan genelde çekinilir ancak unit test ile değişiklik yapılan sınıfın hala çalışıp çalışmadığı anlaşılabilir.Bütün bileşenleri birbirinden bağımsız olarak test etmek ,bakımı daha kolay olan daha kaliteli yazılımlar ortaya çıkarmamıza ortam hazırlar.

Biz unit testing uygularken functional testingin alt başlıkları olan boundary value ve equivalence partitioning tekniklerini kullandık.

Bu labda da her metodu daha küçük bileşenlerine ayırdık ve bu bileşenlerin her birine unit testing uyguladık.

4 A HIGH LEVEL DESCRIPTION OF FIVE SELECTED TEST CASES YOU HAVE DESIGNED USING COVERAGE INFORMATION, AND HOW THEY HAVE INCREASED CODE COVERAGE

EclEmma toolunda ayrıntılı coverage değerleri bulunmuyordu.Classlardaki metodlar üzerinde test caseler oluşturarak classların genel code covarege oranlarında iyileştirmeler yaptık.

NOT :Range.java ve DataUtilities.java classlarının coverage oranını %100 e çekmek için ekstra test metodları oluşturduk.Bu test classlarının tamamı zipin içindeki klasörde mevcuttur.

Range.java:

1) Equals() metodu için : Range classındaki ‘equals’ metodu için test case oluşturduk.Null bir range değeri ile bir range değeri parametre olarak gönderildiğinde false metodun beklenen çıktısı false olmalıydı.Bunun için yazdığımız test metodu ‘testEqualsForNullObject() ‘ coverage değeri %0.6 lık bir artış gösterdi.
2) hashCode() metodu için : Range classındaki ‘HashCode() ‘ metodu için belirli bir range için beklenen değer hesaplayarak bir test case oluşturduk.Bu metod verilen range nesnesinin hashcodeunu döndürmektedir.Bunu test etmek için oluşturduğumuz ‘testHashCode()’ test metodu ile coverage yaklaşık olarak %8 artış gösterdi.
3) combine() metodu için : Bu metod parametre olarak iki range nesnesi alıyor herhangi birisi null ise her zaman ikinci parametreyi döndürüyor.Değilse iki rangein min,max değerlerini alarak yeni range oluşturup döndürüyor.Biz bu metod için test case oluşturduk.’testCombineForOneNullObject’ burada ikinci parametre null olduğunda ilk parametreyi döndürmesi bekleniyor.Bu metodla bu durumu test ettik.Ve coverage oranı yaklaşık % 0.6 artış gösterdi.
4) Constrain() metodu için: Oluşturduğumuz test caseler parametre olarak giden değerin upper değerden büyük olması durumu ve lower değerden küçük olması durumu bu durumlarda metod class range değerinin upper ve lower değerlerini döndürmeli oluşturduğumuz test metodları ‘testConstrainOutOfRangebigUpper’ ve ‘testConstrainOutOfRangelowLower’ bunlar ile coverage değeri %5 artış gösterdi.
5) expandToInclude() metodu için : Bu metoda ikinci parametre olarak giden range nesnesinin upperından büyükse lower aynı kalıyor upperı bu değere getirip yeni bir range nesnesi döndürüyor.Bu durumu ‘testexpandToIncludeUpper()’ metodu ile test ettik ve coverage değeri %2 artış gösterdi.

DataUtilies.java:

(1)createNumberArray() metodu için: Bu fonksiyon parametre olarak null bir data dizisi aldığında exception fırlatması gerekiyor.Bu bizim test casemizdi bunu doğrulamak için de 'testCreateNumberArrayForNullData()' test metodunu oluşturduk. Coverage oranında %3lük bir artış meydana geldi.

getCumulativePercantages() metodu için : Bu metod için iki test case oluşturduk birincisi parametre olarak verilen datanın null olmaması durumu bunun için oluşturduğumuz test metodu :
(2)'testCumulativePercantagesUsingThreeKeysAndValues' : Bu coveragede %31 artış sağladı.
İkinci metod ise parametre olarak verilen datanın null olması durumu bunun için :
(3) 'testCumulativePercantagesforNullData' : metodunu oluşturduk.Coveragede yaklaşık olarak %2 lik bir artış meydana geldi.

createNumberArray2D() metodu için : Parametre olarak iki boyutlu double değerlerden oluşan bir array alıyor.Oluşturduğumuz test case gelen datanın null olması durumunda beklenen değerinin bir exception fırlatıyor olması. Bunu test etmek için (4)'testCreateNumber2DForNullDataSet' fonksiyonunu oluşturduk.Bu fonksiyon ile coverage değerinde yaklaşık %3 artış meydana geldi.

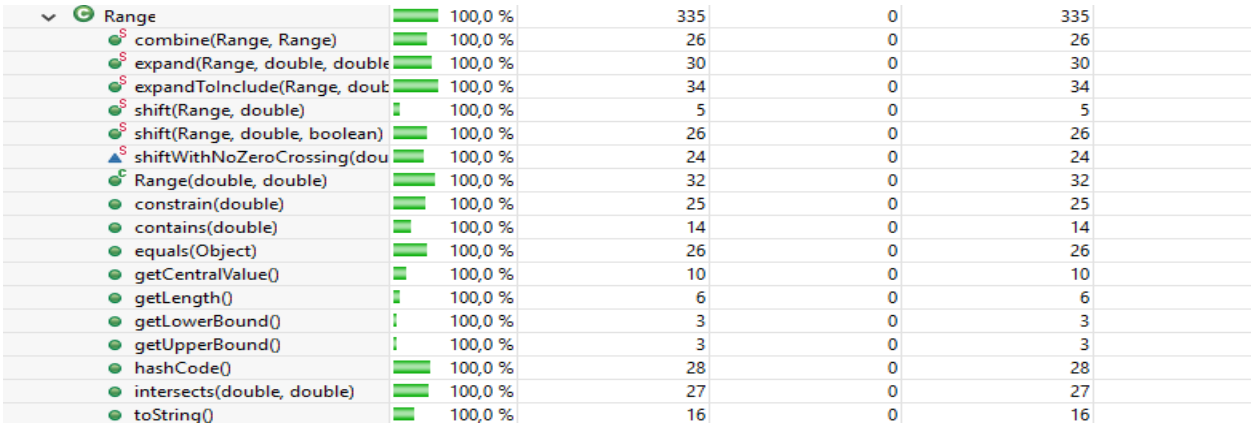
CalculateRowTotalForNullItem(): fonksiyonu için null değerlerden oluşan 2 boyutlu bir tabloda dönüş değerinin 0 olması beklenen bir test case oluşturduk.Bunun için (5)'testCalculateRowTotalForNullItem' test fonksiyonunu meydana getirdik.Test coverageinde herhangi bir değişiklik meydana gelmedi.

DefaultKeyedValues.java:

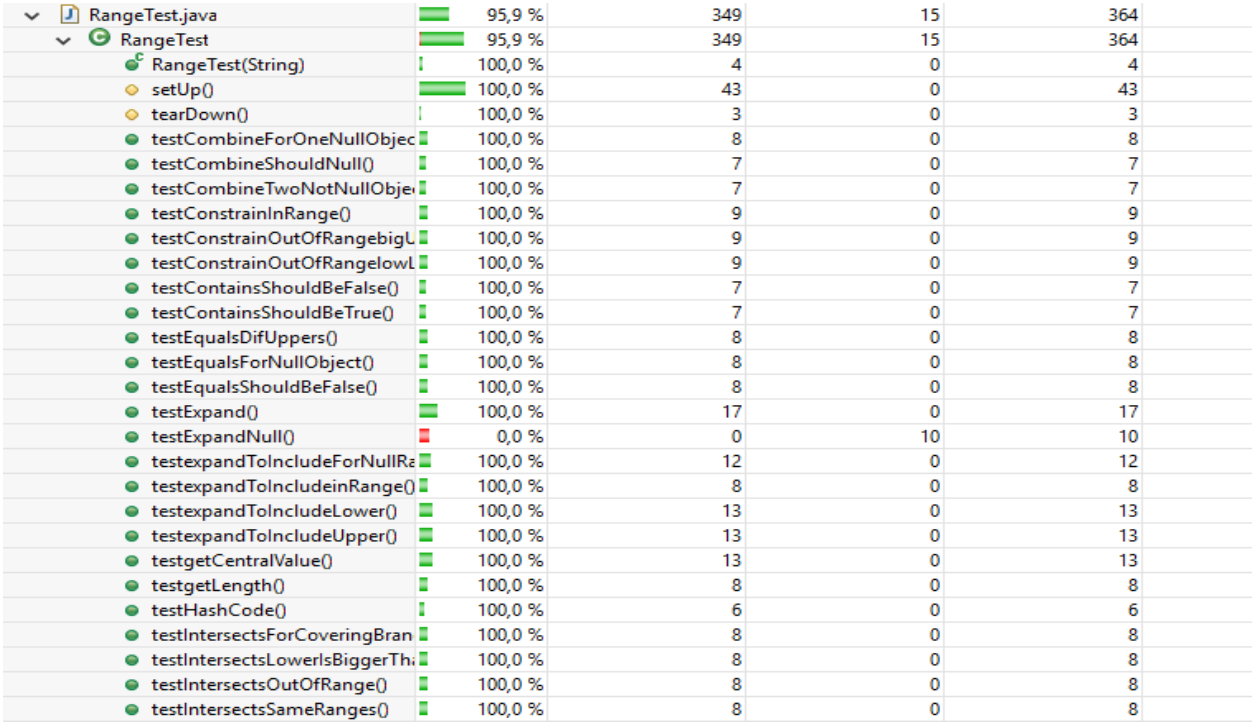
1) 'testEqualsShouldBeTrue()' : İki aynı key ve valuelardan oluşmuş DefaultKeyedValues nesnelerinin 'equals()' metodu ile karşılaştırılmasında beklenen çıktı true olmalıydı. Bu bizim test caseimizdi.Bunu test etmek amacıyla bu metodu oluşturduk ve DefaultKeyedValues classının coveraginde %1 artış meydana geldi.
2) 'testEqualsShouldBeTrue()' : Farklı key ve valueya sahip iki nesne karşılaştırılınca beklenen çıktı false olmalıydı.Bu durumu test etmek için bu metodu oluşturduk ve coverage değerinde %0.7 artış meydana geldi.
3) 'testEqualsForSameObject() ': 'equals' metodu bir DefaultKEyedValues nesnesi tarafından çağırılıp parametre olarak yine aynı nesne verildiği durumda beklenen çıktı true olmalıydı.Bu test casei doğrulamak amacıyla bu metodu oluşturduk.Bu metod ile coverage oranında %0.7 artış meydana geldi.
4) 'testHashCode()' : DefaultKeyedValues sınıfındaki HashCode metodu verilen nesnesinin datasının hashCodunu döndürmektedir'testHashCode()' test metodunda bunun testi için Object.hashCode() metodundan faydalandık ,assertEquals ile gelen değeri kaşılaştırarak test işlemimizi tamamladık.Coverage oranı yaklaşık olarak %2 artış gösterdi.
5) 'testGetKeys()': DefaultKeyedValues sınıfındaki 'getKeys()' metodunu test etmek için bu metodu kullandık.Bu metod nesnenin keylerini döndürmektedir.Bu test metodu coverage oranında yaklaşık olarak %7 artışa sebep oldu.

5 A DETAILED REPORT OF THE COVERAGE ACHIEVED OF EACH CLASS AND METHOD (A SCREEN SHOT FROM THE CODE COVER RESULTS IN GREEN AND RED COLOR WOULD SUFFICE)

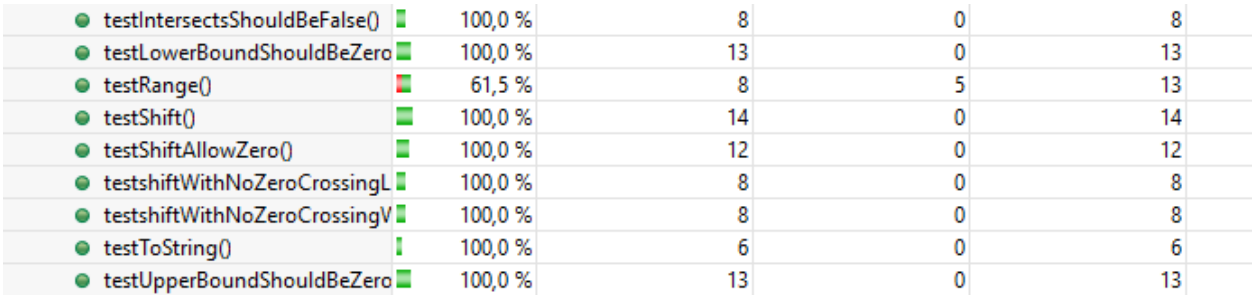
For Range class:



Range	100,0 %	335	0	335
combine(Range, Range)	100,0 %	26	0	26
expand(Range, double, double	100,0 %	30	0	30
expandToInclude(Range, doub	100,0 %	34	0	34
shift(Range, double)	100,0 %	5	0	5
shift(Range, double, boolean)	100,0 %	26	0	26
shiftWithNoZeroCrossing(dou	100,0 %	24	0	24
Range(double, double)	100,0 %	32	0	32
constrain(double)	100,0 %	25	0	25
contains(double)	100,0 %	14	0	14
equals(Object)	100,0 %	26	0	26
getCentralValue()	100,0 %	10	0	10
getLength()	100,0 %	6	0	6
getLowerBound()	100,0 %	3	0	3
getUpperBound()	100,0 %	3	0	3
hashCode()	100,0 %	28	0	28
intersects(double, double)	100,0 %	27	0	27
toString()	100,0 %	16	0	16



RangeTest.java	95,9 %	349	15	364
RangeTest	95,9 %	349	15	364
RangeTest(String)	100,0 %	4	0	4
setUp()	100,0 %	43	0	43
tearDown()	100,0 %	3	0	3
testCombineForOneNullObjec	100,0 %	8	0	8
testCombineShouldNull()	100,0 %	7	0	7
testCombineTwoNotNullObje	100,0 %	7	0	7
testConstrainInRange()	100,0 %	9	0	9
testConstrainOutOfRangebigL	100,0 %	9	0	9
testConstrainOutOfRangelowL	100,0 %	9	0	9
testContainsShouldBeFalse()	100,0 %	7	0	7
testContainsShouldBeTrue()	100,0 %	7	0	7
testEqualsDifUppers()	100,0 %	8	0	8
testEqualsForNullObject()	100,0 %	8	0	8
testEqualsShouldBeFalse()	100,0 %	8	0	8
testExpand()	100,0 %	17	0	17
testExpandNull()	0,0 %	0	10	10
testexpandToIncludeForNullRa	100,0 %	12	0	12
testexpandToIncludeinRange()	100,0 %	8	0	8
testexpandToIncludeLower()	100,0 %	13	0	13
testexpandToIncludeUpper()	100,0 %	13	0	13
testgetCentralValue()	100,0 %	13	0	13
testgetLength()	100,0 %	8	0	8
testHashCode()	100,0 %	6	0	6
testIntersectsForCoveringBran	100,0 %	8	0	8
testIntersectsLowerIsBiggerTh	100,0 %	8	0	8
testIntersectsOutOfRange()	100,0 %	8	0	8
testIntersectsSameRanges()	100,0 %	8	0	8



testIntersectsShouldBeFalse()	100,0 %	8	0	8
testLowerBoundShouldBeZero	100,0 %	13	0	13
testRange()	61,5 %	8	5	13
testShift()	100,0 %	14	0	14
testShiftAllowZero()	100,0 %	12	0	12
testshiftWithNoZeroCrossingL	100,0 %	8	0	8
testshiftWithNoZeroCrossingV	100,0 %	8	0	8
testToString()	100,0 %	6	0	6
testUpperBoundShouldBeZero	100,0 %	13	0	13

For DataUtilities:

DataUtilities.java	98,4 %	179	3	182
DataUtilities	98,4 %	179	3	182
calculateColumnTotal(Values2	100,0 %	26	0	26
calculateRowTotal(Values2D, ir	100,0 %	26	0	26
createNumberArray(double[])	100,0 %	30	0	30
createNumberArray2D(double	100,0 %	29	0	29
getCumulativePercentages(Ke	100,0 %	68	0	68
DataUtilitiesTest.java	96,3 %	312	12	324
DataUtilitiesTest	95,0 %	226	12	238
setUp()	100,0 %	3	0	3
tearDown()	100,0 %	3	0	3
testCalculateColumnTotalForT	100,0 %	25	0	25
testCalculateRowTotalForNullI	100,0 %	27	0	27
testCalculateRowTotalForTwoV	100,0 %	25	0	25
testCreateNumberArray2DForI	60,0 %	6	4	10
testCreateNumberArray2DFor	100,0 %	65	0	65
testCreateNumberArrayForNu	60,0 %	6	4	10
testCreateNumberArrayForTw	100,0 %	27	0	27
testCumulativePercantagesfor	66,7 %	8	4	12
testCumulativePercantagesUsi	100,0 %	28	0	28

For DefaultKeyedValues :

DefaultKeyedValues.java	60,6 %	180	117	297
DefaultKeyedValues	60,6 %	180	117	297
DefaultKeyedValues()	100,0 %	8	0	8
addValue(Comparable, double	100,0 %	8	0	8
addValue(Comparable, Numb	100,0 %	5	0	5
clone()	0,0 %	0	12	12
equals(Object)	84,6 %	55	10	65
getData()	100,0 %	3	0	3
getIndex(Comparable)	50,0 %	12	12	24
getItemCount()	100,0 %	4	0	4
getKey(int)	100,0 %	15	0	15
getKeys()	100,0 %	23	0	23
getValue(Comparable)	0,0 %	0	21	21
getValue(int)	100,0 %	15	0	15
hashCode()	88,9 %	8	1	9
removeValue(Comparable)	0,0 %	0	10	10
removeValue(int)	0,0 %	0	6	6
setData(List)	100,0 %	4	0	4
setValue(Comparable, double)	0,0 %	0	8	8
setValue(Comparable, Numbe	57,1 %	20	15	35
sortByKeys(SortOrder)	0,0 %	0	11	11
sortByValues(SortOrder)	0,0 %	0	11	11

DefaultKeyedValuesTest.java	100,0 %	118	0	118
DefaultKeyedValuesTest	100,0 %	118	0	118
DefaultKeyedValuesTest(String	100,0 %	4	0	4
setUp()	100,0 %	3	0	3
tearDown()	100,0 %	3	0	3
testEqualsForSameObject()	100,0 %	18	0	18
testEqualsShouldBeFalseForDif	100,0 %	31	0	31
testEqualsShouldBeTrue()	100,0 %	31	0	31
testGetKeys()	100,0 %	17	0	17
testHashCode()	100,0 %	11	0	11

6 A COMPARISON ON THE ADVANTAGES AND DISADVANTAGES OF REQUIREMENTS-BASED TEST GENERATION AND COVERAGE-BASED TEST GENERATION.

İkinci assignmentta requirements-based test generation yapmıştık.Bu testi yaparken test caseleri gereksinimlere göre türetmiştik.Metodların işleyişi hakkında fikrimiz olmadığından da gözümüzden kaçan test caseler olabilirdi,yani etkili bir şekilde test caseler üretemeyebilirdik.Yine metodlarda oluşabilecek hata durumlarını kaynak koda erişemediğimizden tahmin edemedik.,bazı branch,statementlar için gerekli testleri yapamadık.Üretebildiğimiz test caseler kodun küçük bir kısmını kapsıyordu yani yeterli değildi.

Bu assignmentta ise coverage-based test generation yöntemini kullandık.Burada white-box testing yaptığımız için, kaynak kod üzerinden bütün olası durumlar için bir test case oluşturduk.Bir test metodu başarısız olduğunda kaynak koda bakarak sebebini kolayca bulabildik.Yani kısacası requirement-based testinge göre daha etkili ve çok daha fazla test case oluşturmuş olduk.Ancak coverage-based ın diğerine göre tek eksi yönü koddaki bütün metodları satır satır anlamanın bunlara göre test case üretmenin biraz fazla zaman alması.

7 A DISCUSSION ON HOW THE TEAM WORK/EFFORT WAS DIVIDED AND MANAGED

Code coverage oranlarını yükseltmek için test caseler oluştururken ikimiz birlikte öncelikle javadocu inceledik,daha sonra beraber kodu inceleyip daha hızlı bir şekilde farklı birimler için test caseler oluşturduk.Laboratuvarın en son parçası olan data-flow coverage in manual olarak hesaplanması kısmında da birimiz control-flow grafiğini çizerken diğerimiz ondan çıkarım yaparak tabloları oluşturdu.Ve son olarak beraber coverage oranını hesapladık.

8 ANY DIFFICULTIES ENCOUNTERED, CHALLENGES OVERCOME, AND LESSONS LEARNED FROM PERFORMING THE LAB

Bu assignmentta föyde verilen test coverage toollarından birçoğu kullandığımız Eclipse versiyonu ile uyumlu değildi, bunun için biraz araştırma yaparak uygun eklenti olarak eclEmmayı bulduk.Eklentiyi kurma sorunumuzdan sonra istenen sınıflar için test caseler oluşturduk ve whitebox testing uyguladık.Zorlandığımız bir diğer konu ise test edilecek sınıftaki bazı metodları anlamak oldu.Bu problem için de jfree.org daki javadoc bize yardımcı oldu.Son olarak da data-flow covaragein manual olarak hesaplanmasında önceden bilgi sahibi olmadığımız için biraz zorlandık.

Bu assignmentta Eclipse üzerinde white box testing uygulanmasını öğrendik ve Eclipse in test toolları hakkında bilgi sahibi olmuş olduk.

9 COMMENTS/FEEDBACK ON THE LAB ITSELF

Bu assignmentta white box testingin yorucu ancak daha eğlenceli olduğunu farkettik.Coverage-based test yöntemi ile requirement basede göre daha etkin test caseler oluşturabileceğimizi anladık.Ancak diğer lablara göre çok daha uğraştırıcı bir ödev oldu bizim için. Ayrıca Eclipse eklentisi olan eclEmmanın kullanıcı arayüzü code coveragei anlamak açından gayet anlaşılırdı.