Assignment 1

INF3121

v15

Nicklas Mortensen Hamang (nicklash)
Huy Ba Nguyen (huybn)
Dana Zangana (danaaz)

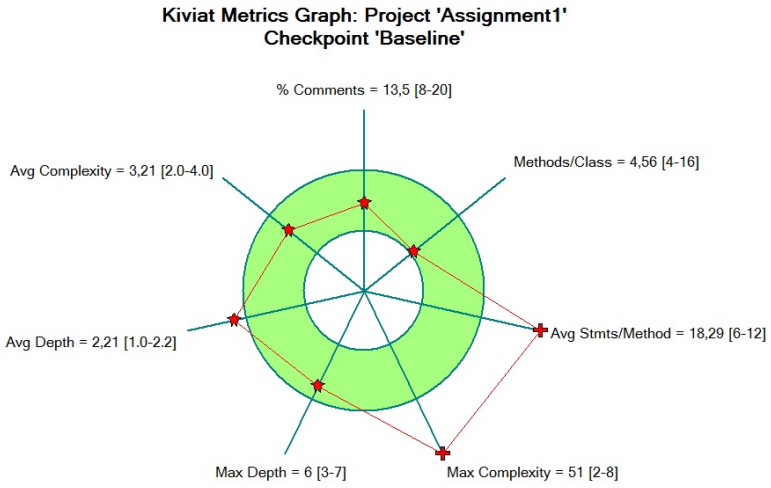
We have chosen to use part 2 of the project assignment of INF2100 called AlboC. it is all stored at an online repository on the site github. If the screenshot appear to small they to are on the repository.

Link provided here: (https://github.com/Eckename/INF3121/tree/master/no/uio/ifi)
Screenshots: (https://github.com/Eckename/INF3121/tree/master/Screenshot/Used)

3 a)

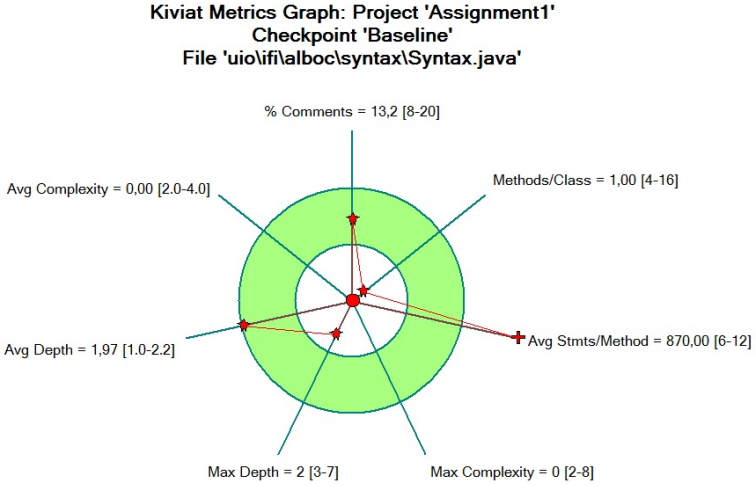
The graph tells us that we have an acceptable amount of comments in relation to the size of the code we have chosen. The average complexity, max depth and the method per class follows suit in regards of being within the acceptable limits. The areas that need improvement (not acceptable) are shown in our graph to be the average depth, max complexity and average statement per method.

We interpret the metrics by looking at its value in relation to the value's accepted range. i.e our comments are given a value of 13.5 and the range is 8-20.



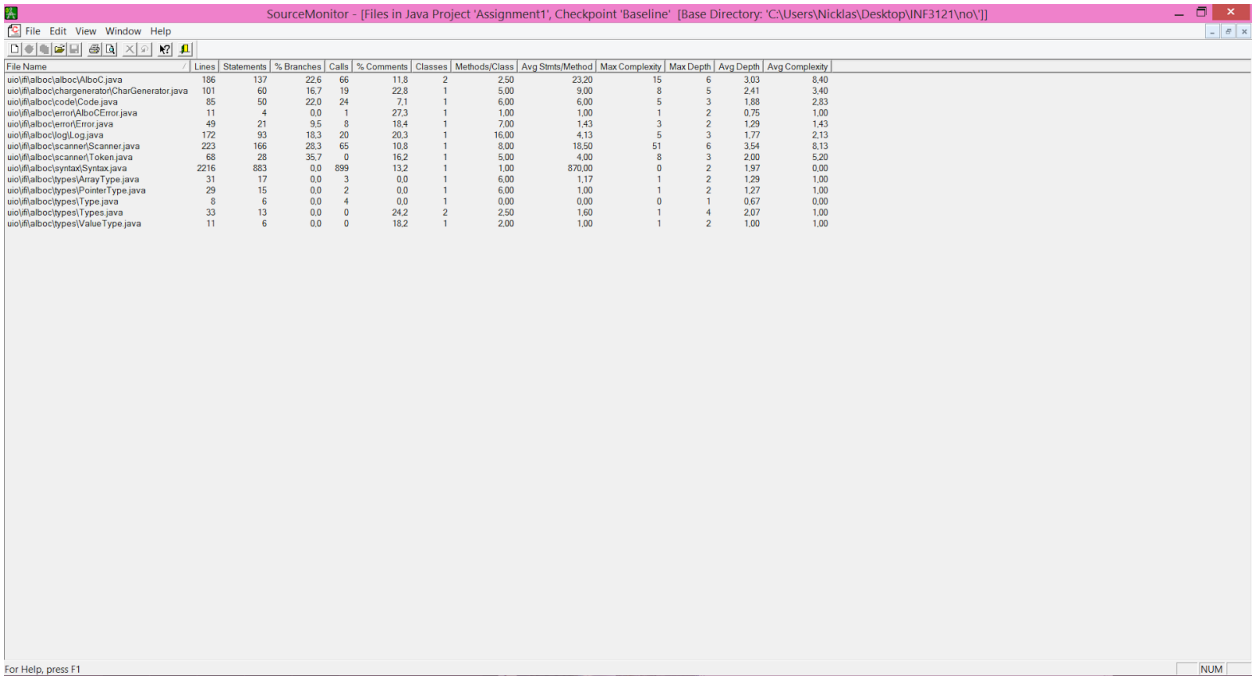
3 b)
Again the graph tell us what need improvement and what does not, in the case of the file Syntax.java in our project, which is methods per class, the average statments per method and its max depth. This can help us determine where to make the improvements that are needed. First seen in comparison to the kiviat graph for the entire project is the lack of a complexity metric. It also shows that the files average

statement per method is exceedingly higher, the method per class is somewhat lower, as is the max depth and the average depth is moved inside the green “doughnut”.



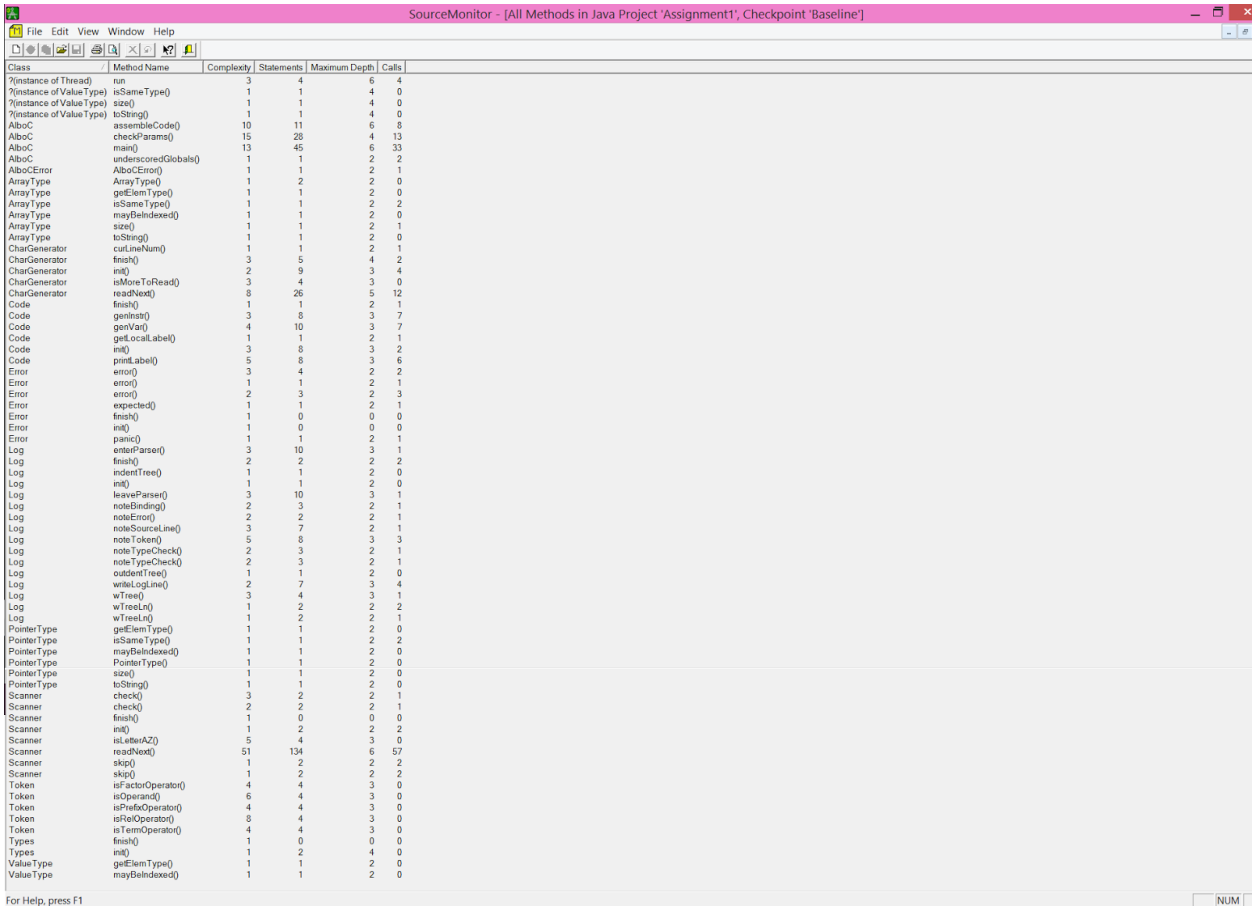
4)

The biggest file in relation to lines in our archive is the file Syntax.java which contains 2216 lines and an average of 1 method over it’s 1 class.



5)

The most complex method in our entire project is within the file/class Scanner by the name of readNext(). It has the complexity of 51 and contains 134 statements. I would refactor many of the methods in this project, like aforementioned method and it's class. This program is quite coupled as many of it's classes calls on methods within other classes and even other files.



Class	Method Name	Complexity	Statements	Maximum Depth	Calls
?(instance of Thread)	run	3	4	6	4
?(instance of ValueType)	isSameType()	1	1	4	0
?(instance of ValueType)	size()	1	1	4	0
?(instance of ValueType)	toString()	1	1	4	0
AlboC	assembleCode()	10	11	6	8
AlboC	checkParams()	15	28	4	13
AlboC	main()	13	45	6	33
AlboC	underscoredGlobals()	1	1	2	2
AlboCError	AlboCError()	1	1	2	1
ArrayType	ArrayType()	1	2	2	0
ArrayType	getElemType()	1	1	2	0
ArrayType	isSameType()	1	1	2	2
ArrayType	mayBeIndexed()	1	1	2	0
ArrayType	size()	1	1	2	1
ArrayType	toString()	1	1	2	0
CharGenerator	curLineNum()	1	1	2	1
CharGenerator	finish()	3	5	4	2
CharGenerator	init()	2	9	3	4
CharGenerator	isMoreToRead()	3	4	3	0
CharGenerator	readNext()	8	26	5	12
Code	finish()	1	1	2	1
Code	genInstr()	3	8	3	7
Code	genVar()	4	10	3	7
Code	getLocalLabel()	1	1	2	1
Code	init()	3	8	3	2
Code	printLabel()	5	8	3	6
Error	error()	3	4	2	2
Error	error()	1	1	2	1
Error	error()	2	3	2	3
Error	expected()	1	1	2	1
Error	finish()	1	0	0	0
Error	init()	1	0	0	0
Error	panic()	1	1	2	1
Log	enterParser()	3	10	3	1
Log	finish()	2	2	2	2
Log	indentTree()	1	1	2	0
Log	init()	1	1	2	0
Log	leaveParser()	3	10	3	1
Log	noteBinding()	2	3	2	1
Log	noteError()	2	2	2	1
Log	noteSourceLine()	3	7	2	1
Log	noteToken()	5	8	3	3
Log	noteTypeCheck()	2	3	2	1
Log	noteTypeCheck()	2	3	2	1
Log	outdentTree()	1	1	2	0
Log	writeLogLine()	2	7	3	4
Log	wTree()	3	4	3	1
Log	wTreeLn()	1	2	2	2
Log	wTreeLn()	1	2	2	1
PointerType	getElemType()	1	1	2	0
PointerType	isSameType()	1	1	2	2
PointerType	mayBeIndexed()	1	1	2	0
PointerType	PointerType()	1	1	2	0
PointerType	size()	1	1	2	0
PointerType	toString()	1	1	2	0
Scanner	check()	3	2	2	1
Scanner	check()	2	2	2	1
Scanner	finish()	1	0	0	0
Scanner	init()	1	2	2	2
Scanner	isLetterAZ()	5	4	3	0
Scanner	readNext()	51	134	6	57
Scanner	skip()	1	2	2	2
Scanner	skip()	1	2	2	2
Token	isFactorOperator()	4	4	3	0
Token	isOperand()	6	4	3	0
Token	isPrefixOperator()	4	4	3	0
Token	isRelOperator()	8	4	3	0
Token	isTermOperator()	4	4	3	0
Types	finish()	1	0	0	0
Types	init()	1	2	4	0
ValueType	getElemType()	1	1	2	0
ValueType	mayBeIndexed()	1	1	2	0

6)

The main improvement i would do to my code is to lessen its complexity, probably by reevaluating some of the arithmetics or logical statements in the testing progress, if possible without losing function. Even though the comments fall perfectly within limits I would also like to do some extensions there. Whilst method per class is within the limits it is barely so and could use some looking into.

There are then some metric in single files whose improvement could fix the entire project, like Syntax average statements per class and Scanner's complexity.